

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

«На правах рукопису»
УДК _____

До захисту допущено:
Завідувач кафедри
_____ Сергій СТИПЕНКО
«___» _____ 20__ р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

зі спеціальності 123 «Комп'ютерна інженерія»

на тему: «Система управління портфоліо ІТ-компанії для залучення нових клієнтів»

Виконав (-ла):

студент (-ка) VI курсу, групи ІО-11мп

Петрик Артур Сергійович _____

Науковий керівник:

проф., д.т.н., проф.,

Кулаков Юрій Олексійович _____

Консультант:

проф., д.т.н., проф.,

Жабін Валерій Іванович _____

Рецензент:

доц. каф. ІСТ, к.т.н., доц.,

Коган Алла Вікторівна _____

Засвідчую, що у цій магістерській
дисертації немає запозичень з праць
інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2022 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – другий (магістерський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТИРЕНКО

«___» _____ 20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Петрику Артуру Сергійовичу

1. Тема дисертації «Система управління портфоліо ІТ-компанії для залучення нових клієнтів», науковий керівник дисертації Кулаков Юрій Олексійович, д.т.н., проф., затверджені наказом по університету від «___» _____ 20__ р. № _____

2. Термін подання студентом дисертації: _____

3. Об'єкт дослідження: Цифровий маркетинг та залучення інвестицій

4. Вихідні дані: _____

5. Перелік завдань, які потрібно розробити: огляд існуючих рішень, опис предметної області, вибір технологій для реалізації поставленого завдання, розробка програмного забезпечення

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: загальнотематичні ілюстрації, структурна схема роботи застосунку

7. Орієнтовний перелік публікацій: _____

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормоконтроль	проф. Кулаков Ю.О.		

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Огляд існуючих рішень	05.09.2022 - 11.09.2022	
2	Планування розробки	12.09.2022 - 18.09.2022	
3	Вибір технологій для розробки	18.09.2022 - 21.09.2022	
4	Реалізація системи	22.09.2022 - 30.10.2022	
5	Тестування системи	31.10.2022 - 05.11.2022	
6	Розрахунки стартапу	06.11.2022 - 12.11.2022	
7	Оформлення документації	13.11.2022 - 30.11.2022	

Студент

Петрик Артур Сергійович

Науковий керівник

Кулаков Юрій Олексійович

РЕФЕРАТ

Робота складається із вступу та чотирьох розділів. Загальний обсяг роботи: 103 аркуші основного тексту, 32 рисунки та 23 таблиці. При написанні магістерської роботи було використано 32 різних джерела.

Актуальність проблеми. Хоча багато людей вважають, що мати портфоліо необхідно лише художникам і дизайнерам, незалежно від професії портфоліо є ефективним і недорогим способом продемонструвати цілісне уявлення про навички та конкретні результати попередньої роботи. Коли з'являється можливість, ви хочете бути впевненими, що можете показати всі свої професійні досягнення в красивій і зрозумілій формі. Саме тому для кар'єрного зростання важливо мати готове цифрове портфоліо на будь-який випадок.

Мета і завдання дослідження. Завданням роботи є дослідження можливостей залучення нової клієнтської аудиторії для ІТ-компанії шляхом розповсюдження спеціалізованих портфоліо. Метою магістерської роботи є розроблення системи менеджменту портфоліо власних проектів з метою розповсюдження та збільшення кількості зацікавлених клієнтів.

Об'єкт дослідження. Цифровий маркетинг та залучення інвестицій.

Предмет дослідження. Система, що дозволить адаптувати інструментарій по управлінню портфоліо компанії для залучення нових клієнтів та потенційних інвестицій.

Практична цінність. Розроблений проект є повноцінною системою, що допоможе завжди тримати портфоліо виконаних робіт в актуальному стані, а також допоможе в розповсюдженні власних досліджень серед клієнтських мас.

Ключові слова: сервіс, веб-застосунок, портфоліо.

ABSTRACT

The work consists of an introduction and four sections. The total volume of work: 103 pages of the main text, 32 figures and 23 tables. 32 different sources were used when writing the master's thesis.

The urgency of the problem. Although many people think that having a portfolio is only necessary for artists and designers, regardless of the profession, a portfolio is an effective and inexpensive way to demonstrate a holistic view of skills and specific results of previous work. When the opportunity arises, you want to be sure that you can display all of your professional achievements in a beautiful and understandable way. That is why it is important for career growth to have a ready-made digital portfolio for any occasion.

The purpose and tasks of the research. The task of the work is to research the possibilities of attracting a new client audience for the IT company through the distribution of specialized portfolios. The goal of the master's thesis is to develop a portfolio management system of own projects in order to distribute and increase the number of interested customers.

Object of study. Digital marketing and investment attraction.

Subject of study. A system that will allow you to adapt the company's portfolio management tools to attract new customers and potential investments.

Practical value. The developed project is a full-fledged system that will help to always keep the portfolio of completed works up-to-date, as well as help in the distribution of own research among the client masses.

Keywords: service, web application, portfolio.

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ	3
ВСТУП	4
РОЗДІЛ 1	6
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ	6
1.1 Опис предметної області.....	6
1.2 Огляд існуючих реалізацій	8
1.2.1 Огляд системи Coroflot.....	8
1.2.2 Огляд системи Flickr.....	10
1.2.3 Огляд системи Entsane.....	12
1.2.4 Огляд системи Crevado.....	14
1.3 Порівняння сервісів.....	16
ВИСНОВКИ ДО РОЗДІЛУ 1	18
РОЗДІЛ 2	19
ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ	19
2.1. Вибір мови програмування	19
2.2. Вибір фреймворку	28
2.2. Вибір сховища даних	35
2.3. Вибір інструменту для виконання відкладених задач.....	39
2.3. Вибір технологій для реалізації веб-інтерфейсу.....	41
ВИСНОВКИ ДО РОЗДІЛУ 2	43
РОЗДІЛ 3	44
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	44

3.1. Структура розробленого застосунку	44
3.2. Побудова веб-сервера за допомогою Django	46
3.3. Робота з базою даних	53
3.4. Інтеграція Elasticsearch	60
3.5. Сервіс для відкладених задач.....	64
3.6. Реалізація клієнтської частини	67
ВИСНОВКИ ДО РОЗДІЛУ 3	73
РОЗДІЛ 4	74
РОЗРОБКА СТАРТАП ПРОЕКТУ	74
4.1. Опис ідеї проекту	74
4.2. Технологічний аудит проекту	75
4.3. Аналіз ринкових можливостей запуску стартап-проекту	76
4.4. Розроблення ринкової стратегії проекту	85
4.5. Розроблення маркетингової програми стартап-проекту	87
ВИСНОВКИ ДО РОЗДІЛУ 4	92
ВИСНОВКИ.....	93
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	95
Додаток А.....	99

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ

API – Application Programming Interface

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

IP – Internet Protocol

IT – Information Technology

JSON – JavaScript Object Notation

JVM – Java Virtual Machine

MVC – Model-View-Controller

MVP – Minimum Viable Product

ORM – Object-Relational Mapping

REST – Representation State Transfer

SQL – Structured Query Language

UI – User Interface

URL – Uniform Resource Locator

UX – User Experience

XML – eXtensible Markup Language

БД – База даних

ОС – операційна система

ПЗ – Програмне забезпечення

СУБД – Система управління базами даних

ВСТУП

Наявність цифрового маркетингового портфолію як ніколи важливо в сучасному діловому світі. Портфолію – це спосіб продемонструвати свої навички, здібності та досвід потенційним роботодавцям і клієнтам.

Потужне цифрове маркетингове портфолію допоможе отримати роботу вашої мрії або значно прискорить залучення клієнтської бази. Це також може бути корисним інструментом для спілкування та побудови стосунків у тій чи іншій індустрії.

Портфолію цифрового маркетингу – це колекція найкращих робіт кандидата. Воно може бути у формі дописів у блозі, статей, інфографіки чи будь-якого іншого вмісту, який ви створили. Це також може включати роботу, яку ви виконали для клієнтів або роботодавців, наприклад, бізнес проекти, кампанії в соціальних мережах, маркетингові ініціативи, що включають електронну комерцію, дизайн, розробку веб-сайтів, проектування комплексних систем, втілення інноваційних ідей чи підтримка вже існуючих рішень.

Цифрове портфолію проектів має точно відображати навички та досвід кандидата. Воно має бути гарно організовано та зручне для навігації, щоб потенційні клієнти могли легко та швидко знайти потрібну інформацію.

Маркетингове портфолію можна уявити як свій особистий додаток, веб-сайт або блог, який демонструє ваші найкращі роботи. Воно ставить перед собою мету показати потенційним клієнтам, які задачі ви можете виконувати і яких результатів ви можете досягти.

Є кілька основних причин, чому вам потрібне цифрове маркетингове портфолію:

- Це допоможе вам знайти роботу та клієнтів. Ваше цифрове маркетингове портфолію схоже на ваше резюме, але з більшою глибиною та креативністю. Це дозволяє вам продемонструвати свої навички та досвід таким способом, який є набагато привабливішим, ніж традиційне резюме. Як наслідок, це може допомогти вам значно легше знайти клієнтуру на

фоні інших претендентів, що надають послуги не маючи власного портфелю попередньо виконаних робіт.

- Це робить вас більш надійним. Цифрове портфоліо – це чудовий спосіб завоювати довіру потенційних роботодавців. Демонстрація їм вашої попередньої роботи, а також досягнутих результатів дозволить їм побачити ваші здібності та рівень професійності.
- Це виділяє вас серед конкурентів. З такою кількістю людей, які змагаються за роботу, важливо знайти способи виділити себе з натовпу. Цифрове портфоліо – чудовий спосіб зробити це, адже не в кожного воно є.

Завданням цієї магістерської роботи є розробка системи управління портфелю власних проєктів для використання саме працівниками сучасних айті корпорацій. Адже відомою стратегією цифрового маркетингу є правильний менеджмент та розповсюдження власного портфоліо. Зазвичай, компанії, які не тримають це питання в актуальному стані, ведуть портфоліо своїх проєктів у гугл таблицях, excel файлах чи в чомусь подібному. Однією із задач є автоматизація цього процесу та перетворення таких таблиць в більш презентабельний вигляд для кінцевого клієнта.

Цільовою аудиторією даного проєкту є відділ продажів або маркетингу будь-якої сервісної ІТ компанії.

Даний інструмент націлений на допомогу в двох ключових питаннях: завжди тримати власне портфоліо в актуальному стані, а також здійснювати допомогу при пошуку нових клієнтів.

РОЗДІЛ 1

ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Опис предметної області

Маркетингове портфоліо – це веб-сайт або веб-сторінка, яка демонструє найкращі проекти потенційним роботодавцям або клієнтам. Портфоліо має містити короткий опис професійного досвіду та включати принаймні декілька сильних тематичних досліджень, що демонструють професійні навички [1]. Сам веб-сайт також є частиною портфоліо, яка демонструє вміння використовувати цифрові технології та створювати унікальні стилі.

Питання, яке часто виникає, полягає в тому, чи можна включити всі клієнтські проекти у своє маркетингове портфоліо. Відповідь на це запитання залежить від кількох факторів [2]. Прикладом може бути договір про нерозголошення, який ви укладаєте з клієнтом або ж дозволений рівень розповсюдження, якщо ви ділитесь інформацією про той чи інший проект. У деяких випадках може бути цілком нормальним продемонструвати таку роботу для наступного клієнта у своєму портфоліо. Наприклад, якщо ви маєте постійний контракт із замовником або якщо його влаштовує той варіант, щоб публічно ділилися розробкою. Однак, бувають випадки, коли демонструвати роботу клієнта у вашому портфоліо недоцільно. Наприклад, якщо у вас немає дозволу та замовник бажає зберегти свою роботу приватною.

Крім того, якщо ви спеціалізуєтесь в певній галузі цифрового маркетингу (наприклад, маркетинг у соціальних мережах або маркетинг електронною поштою), може бути корисно продемонструвати свою роботу в цій галузі. Це дасть потенційним клієнтам краще уявлення про тип роботи, яку ви виконуєте, а також характеристики й можливості кінцевого продукту.

Можна подумати, що портфоліо призначені лише для дизайнерів, і звичайно, можна черпати натхнення з портфоліо дизайну UX та UI, коли справа доходить до створення власного, але вони не менш важливі в сфері маркетингу. Незалежно від того, чи ви початковий дизайнер чи маркетолог відомої ІТ-

корпорації, кінцева мета однакова: продемонструвати потенційним клієнтам та роботодавцям свою цінність, навички та досвід.

Менеджмент портфоліо – це механізм, призначений для трансляції стратегії в портфель проектів для подальшої реалізації, планування, аналізу та переоцінки портфоліо з метою ефективного досягнення стратегічних цілей організації [3].

Система управління портфоліо ІТ-компанії для залучення нових клієнтів – це програмне забезпечення для реалізації операцій доповнення, редагування, фільтрування, пошуку виконаних компанією проектів тими чи іншими сучасними технологіями у відповідній сфері надання послуг для подальшого поширення цього портфоліо з метою зацікавлення нових клієнтів для реалізації їх інвестиційних цілей, а також для створення власних спеціалізованих наборів для подальшого розповсюдження серед відповідних клієнтських аудиторій.

Це повноцінна платформа для досліджень та оглядів, яка допомагає покупцям програмного забезпечення та шукачам послуг вибрати найкращого та найбільш підходящого виконавця для реалізації програмного продукту. У той же час це допомагає ІТ-компаніям і постачальникам програмного забезпечення підвищити статистику залучення користувачів, збільшити впливову частку ринку та покращити впізнаваність бренду.

Переваги системи управління проектами:

- для клієнта це швидкий та спрощений процес пошуку найкращого постачальника послуг;
- завжди актуальна та оновлена картина запланованих та поточних завдань компанії;
- наявність професійної аналітики виконавця;
- можливість оперативно та аргументовано впроваджувати чи видаляти проекти;
- ефективний аналіз портфоліо та оцінка потенційних ризиків.

Метою роботи є розробка комплексного програмного продукту, спрямованого на маркетинг на сейлз відділі айті компанії, що дозволить

автоматизувати ведення актуального портфолію, значно спростити його менеджмент та підвищити ефективність залучення нової клієнтської бази для надання професійних послуг в галузі проектування, розробки, тестування та супроводу програмних комплексів.

1.2 Огляд існуючих реалізацій

Зазвичай, архітектура подібних аналогів являє собою рішення, що базується на веб-застосуванні та охоплює бізнес логіку з предметної області застосунку.

Розглянемо існуючі сервіси, функціонал котрих надає можливість менеджменту власного портфелю проектів, ставить перед собою мету рекламувати свої роботи, а також зосереджується на залученні уваги потенційних роботодавців чи клієнтської бази.

1.2.1 Огляд системи Coroflot

Coroflot – це онлайн ресурс для розміщення дизайнерських робіт, який об'єднує роботодавців із креативними професіоналами в різних галузях, включаючи архітектуру, ілюстрації, графіку, 3D моделювання та дизайн. Розробники цього продукту прагнуть надати дизайнерам чудові можливості для кар'єрного зростання, забезпечити інноваційним компаніям доступ до портфолію провідних талантів світу а також відкрити дивовижні дизайнерські роботи глобальній спільноті.

Під час реєстрації веб-застосунок одразу надає можливість вказати професійну інформацію, а саме: посаду чи роль в команді, індустрії, в яких кандидат спеціалізується та може надавати послуги, навички якими володіє для тієї чи іншої розробки, а також локацію перебування у разі, якщо клієнт шукає виконавця з конкретної країни.

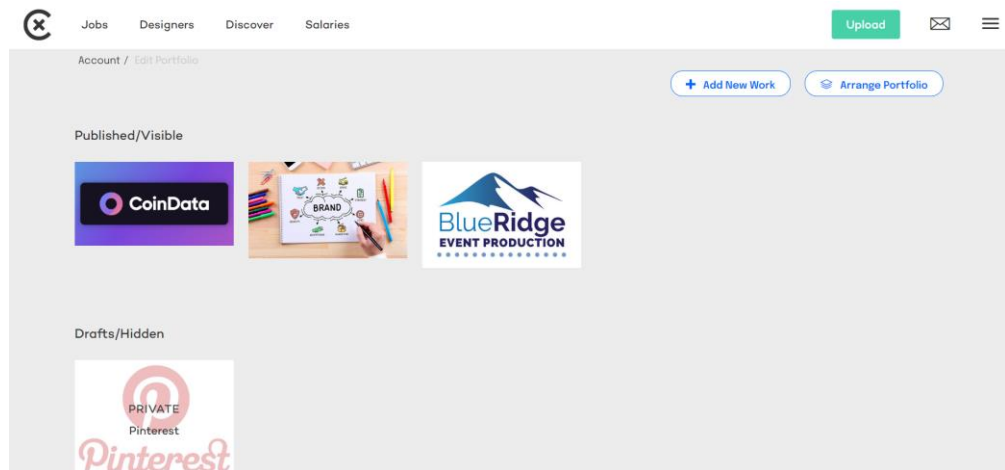


Рис. 1.1 – Управління власним портфоліо на Coroflot

Сервіс надає можливість зберігання приватних проектів чи досліджень у портфоліо (рисунок 1.1). Присутня можливість детального опису усіх робіт, хоча деякі додаткові можливості відкриваються лише із платною підпискою. Користувачу надається можливість упорядкувати портфель власних проектів, якщо на це є необхідність.

Приклад доповнення портфоліо новим проектом представлений на рисунку 1.2.

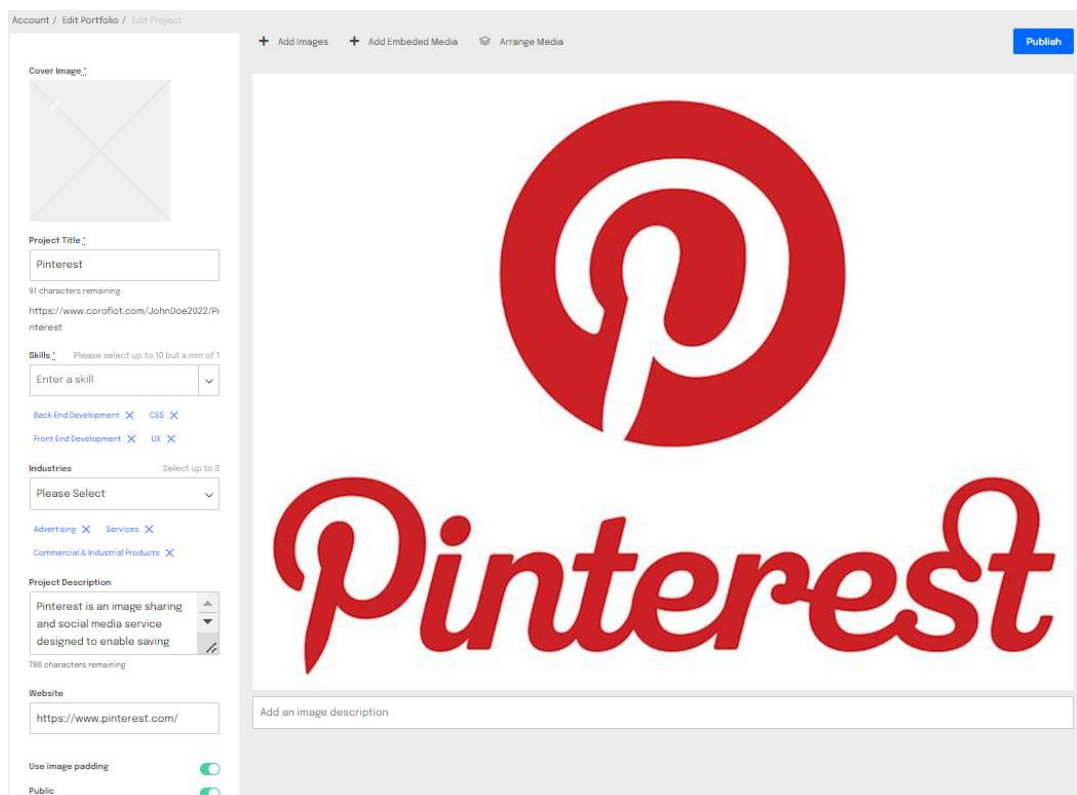


Рис. 1.2 – Додавання нового проекту до портфоліо Coroflot.

Переваги

- впізнаваність бренду: платформа існує понад 20 років та серед своїх клієнтів має таких корпоративних гігантів як Microsoft, Landor, Blu Dot, Nokia, Intel, Nike [4];
- можливість частково змінити посилання на власне портфоліо, наприклад, додати своє ім'я в назву;
- інтуїтивно зрозуміла навігація в системі;
- змога додавання та управління приватними проектами;

Недоліки

- створення проекту в портфоліо без додавання зображень заборонено, а також зображення менші, ніж 440x440 пікселів не підтримуються;
- неможливо додати більше трьох індустрій або ж більше десяти технологій для окремого проекту;
- відсутність функції автоматизованого додавання одразу декількох робіт у власне портфоліо;
- додатковий функціонал пошуку інших виконавців буде доступний лише використовуючи платний режим;
- сервіс не надає можливості фільтру, пошуку та групування певних проектів, для поширення серед особливих клієнтських баз;
- немає можливості поділитися своїм портфоліо через соціальну мережу чи електронну пошту після його перегляду;
- ресурс не здатний остаточно видалити усі дані про користувача.

1.2.2 Огляд системи Flickr

Flickr – це медіа платформа, яка використовується для завантаження, упорядкування та обміну цифровими медіа файлами, такими як фотографії та відео. Її використовують багато професіональних фотографів для надсилання зображень потенційним та постійним клієнтам.

Крім завантаження фотографій, користувачі можуть переглядати публічну стрічку, щоб ознайомитися з роботами інших людей, а також є можливість ставити «лайки» або коментувати все, що викликає інтерес.

Створити обліковий запис на платформі можна безкоштовно, що заохочує навіть звичайних користувачів зареєструватися. Однак для тих, хто користується нею більш серйозно, можна придбати Pro версію [5]. Платна підписка знімає обмеження певних можливостей та розблоковує ряд нових функцій, як-от: розширити сховище завантажуваних файлів; можливість отримувати прогресивну статистику про те, які з робіт переглядаються найчастіше; вимкнути постійну рекламу, що надокучає; отримати пріоритет у вирішенні будь-яких питань із сервісом; включити опцію, яка допомагає швидко та в автоматичному режимі створювати резервні копії усіх колекцій фото чи відео з телефона, комп'ютера, жорсткого диска чи навіть з аккаунтів інших популярних сервісів, наприклад Dropbox, Adobe Lightroom і т.д.

Приклад публічного портфоліо, створеного в сервісі Flickr, показано на рисунку 1.3.

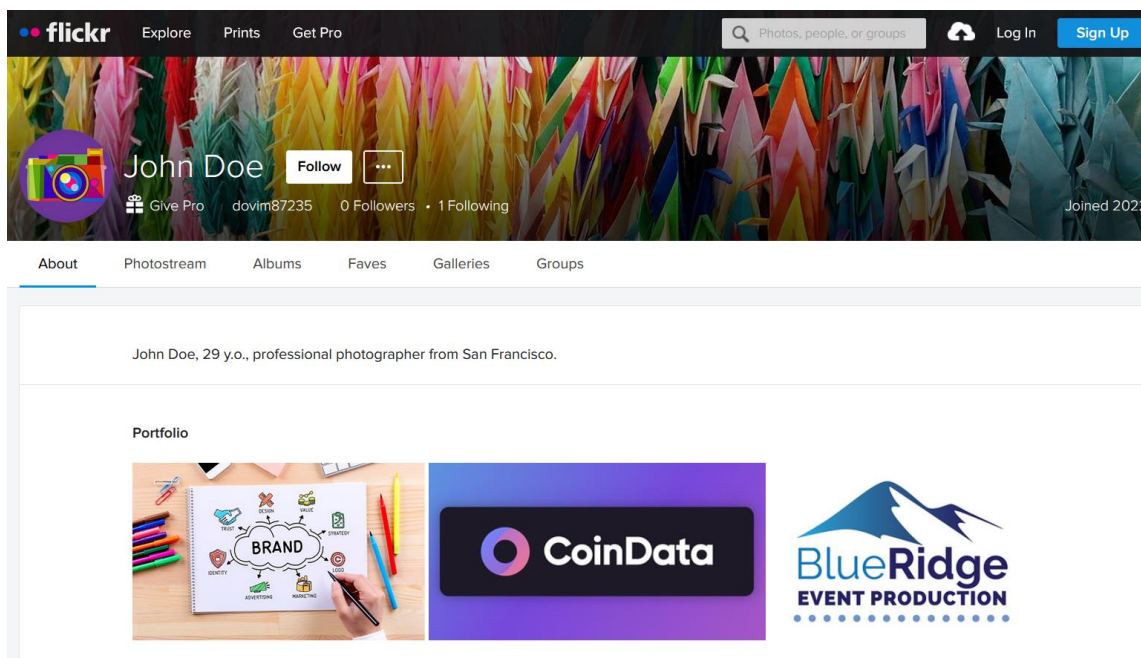


Рис. 1.3 – Загальний вигляд публічного портфоліо з платформи Flickr

Переваги

- наявна функція одночасного додавання чи редагування кількох робіт;

- можливість групувати деякі роботи в альбоми, об'єднуючи їх спільною тематикою для полегшення подальшого пошуку або ж створення окремих альбомів для спеціальних клієнтів, що зацікавлені в конкретній індустрії;
- існують функції пошуку та фільтрації певних робіт;
- надається змога приховати деякі проекти і зробити їх приватними;
- створення резервних копій.

Недоліки

- у безкоштовній версії кількість медіа файлів, які можна завантажити, обмежена до 25;
- неможливість повноцінно описати кожний проекту портфолію, оскільки до завантажених медіа файлів додається лише одне поле «*description*»;
- складний в орієнтуванні інтерфейс, наприклад неможливо переглянути тільки приватні проекти (є лише опція «*View all*») для їх редагування;
- відсутня можливість поділитися власним портфолію або ж окремо визначеною колекцією робіт через соціальну мережу чи електронну пошту;
- немає інтеграції зі сторонніми сервісами для автоматичного входу або оновлення даних профіля (Google, LinkedIn);
- сервіс часто зависає при видаленні власних медіа файлів.

1.2.3 Огляд системи Entsane

Entsane — це веб-застосунок для спілкування, демонстрування та просування власного таланту націлений на фрілансерів та індустрію розваг. Основним призначенням цього сервісу є створення онлайн портфолію та розкриття своїх талантів. Entsane дає доступ до світового ринку та до глобальної бази даних схожих користувачів, з якими можна зв'язатися та започаткувати спільні проекти, а також ділитися досвідом спільноти розваг, що допоможе розвивати та просувати свої таланти чи напрацювання.

Даний сервіс є повністю безкоштовним та ставить перед собою мету допомагати просувати та мотивувати розвиток талантів серед окремих людей або

команд на світовому рівні, заохочувати обмін інформацією, знаннями та досвідом, і налагоджувати співпрацю між міжнародними організаціями, які різними способами сприяють вдосконаленню та постійній модернізації індустрії розваг [6].

Після реєстрації надається можливість заповнити профіль на основі професії користувача для подальшого представлення послуг, які він може надати глобальній спільноті (рисунок 1.4).

Рис. 1.4 – Заповнення профілю та портфолію користувача на Entsane

Переваги

- сервіс повністю безкоштовний;
- легкий у користуванні інтерфейс;
- кросплатформеність;
- обширний набір полів для передання специфіки кандидата.

Недоліки

- доповнення галереї проектів лише по одному зображенню за раз;

- неможливість детального опису характеристик конкретного проекту в єдиному блоці портфоліо;
- нема цілісної сутності проекту/дослідження (неможливість прикріпити зображення з галереї до певного проекту);
- відсутня можливість керування приватними проектами;
- неможливість приховувати або ж регулювати певні роботи в портфоліо для конкретних клієнтів;
- не реалізована функція поширення власного портфоліо клієнтам електронною поштою чи в соц. мережах;
- немає інтеграції зі сторонніми сервісами(Google, LinkedIn...).

1.2.4 Огляд системи Crevado

Crevado – це відомий онлайн застосунок, що полегшує творчим фахівцям створити онлайн портфоліо (рисунок 1.5) для легкої демонстрації своїх робіт в глобальній мережі. Платформою на даний момент користуються більше дев'ятисот тисяч користувачів. За допомогою Crevado також можна продавати свої розробки в Інтернеті – інтеграція PayPal і Fotomoto робить надзвичайно простим отримання онлайн-платежів.

Головна мета сервісу – стати найкращим інструментом, що забезпечує можливості управління портфелем проектів та поширення чи продажу своїх робіт клієнтам з усього світу [7].

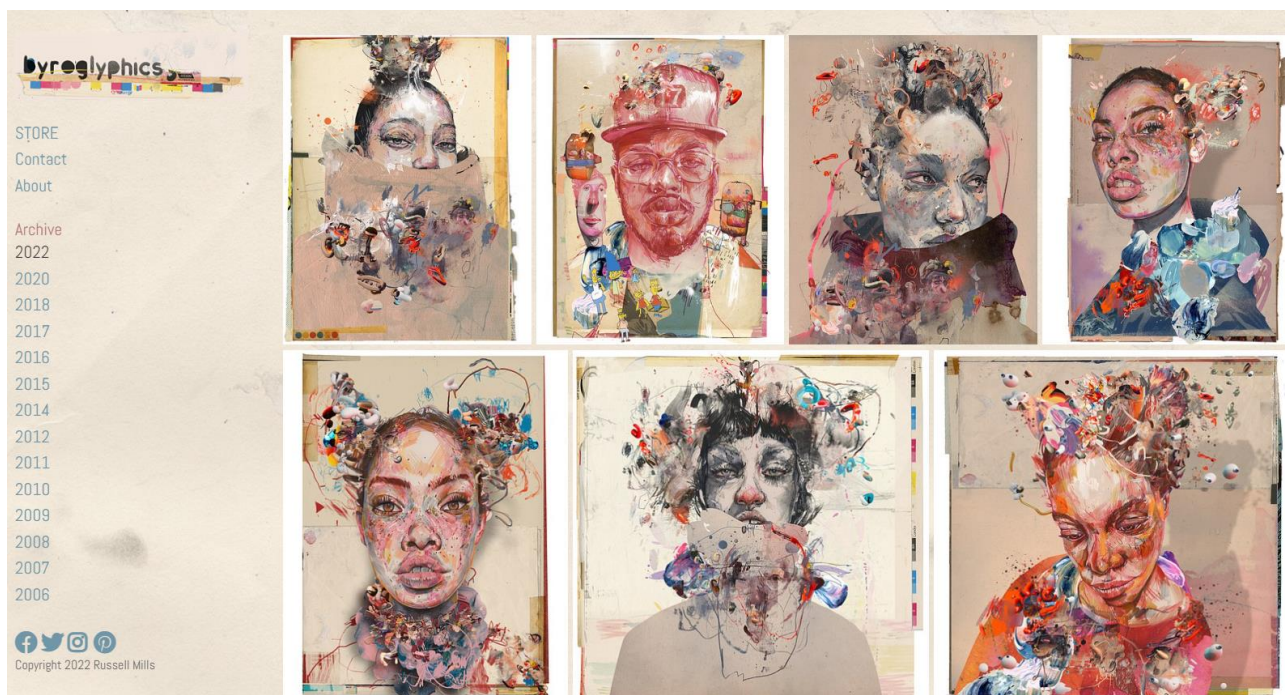


Рис. 1.5 – Портфолію, створене за допомогою сервісу Crevado

Crevado позиціонує себе, як швидкий і простий у використанні додаток: не потрібно бути технічним гуру чи фахівцем у програмуванні для керування власним портфолію. Розробники зазначають, що створені за допомогою даного ресурсу портфолію справляють враження та ідеально підходять для реклами чудових робіт.

Можна приховувати деякі роботи від публічного виставлення та навіть керувати налаштуваннями конфіденційності, встановлюючи паролі на деякі із власних робіт, захистивши їх від сторонніх очей або ж надавши специфічним клієнтам доступ.

Платформа продовжує розвиватись повністю виходячи з відгуків своєї аудиторії користувачів, а отже розглядають усі пропозиції та відгуки з боку юзера.

Незважаючи на велику кількість переваг, слід зазначити, що значна їх частина відкривається лише після придбання платної версії застосунку, що і є найбільшим мінусом сервісу.

Переваги

- можливість створення унікального дизайну сайту з портфелем своїх робіт;

- кроссплатформеність: дизайни портфоліо автоматично адаптуються до мобільних пристроїв;
- зручність керування та навігації;
- наявна функція обмеження доступу до певних робіт, блокуючи їх паролем;
- варіативність зміни посилання на власне портфоліо.

Недоліки

- значно обмежений функціонал та можливості використання в безкоштовній версії застосунку;
- неможливість (на момент огляду) зареєструватися напряму надаючи облікові дані користувача: ресурс видає помилку безпеки «*Please complete the security check*». Була спроба зареєструватися таким чином використовуючи різні браузері, операційні системи та IP адреси;
- невелика кількість типів медіа файлів, які можна завантажити;
- відсутня функція одночасного додавання декількох проектів у портфоліо;
- немає можливості поділитися колекцією своїх робіт для потенційного клієнта за допомогою електронної пошти або ж соціальних мереж.

1.3 Порівняння сервісів

На основі вище розглянутих систем менеджменту та поширення власного портфоліо здійснимо їх порівняльну характеристику (таблиця 1.1).

Таблиця 1.1

Порівняльний аналіз запропонованих рішень

Характеристика	Coroflot	Flickr	Crevado	Entsane
Достатня множина функцій в безкоштовній версії	+	—	—	+
Налаштування посилань на власне портфоліо	+	—	+	—
Зручність керування	+	—	+	+
Кроссплатформеність	—	+	+	+
Впізнаваність бренду	+	+	+	—

Продовження таблиці 1.1

Групування проектів для клієнтів конкретної індустрії	—	+	—	—
Підтримка приватних проектів	+	+	+	—
Можливість автоматизованого додавання/зміни/видалення одразу кількох робіт	—	—	—	—
Зворотний зв'язок	+	+	+	+
Моніторинг аналітики	—	+	+	—
Імпорт/Експорт портфоліо	—	—	—	—
Інтеграція зі сторонніми сервісами	—	—	+	—
Створення резервних копій	—	+	—	—
Опція поширення портфоліо ел. поштою чи в соц. мережах	—	—	—	—

ВИСНОВКИ ДО РОЗДІЛУ 1

У цьому розділі було проведено аналіз та порівняння вже існуючих програмних рішень, що втілюють концепцію створення та управління власним портфоліо тих чи інших робіт із подальшим просуванням власних доробок і навичок, розповсюджуючи колекцію своїх проектів серед різних клієнтів, з метою залучення інвестицій або ж отримання прямих пропозицій у професійній співпраці.

Дослідивши оглянуті сервіси, бачимо, що усі вони базуються на веб застосуванні та мають свої переваги і недоліки. Хоч за структурою платформи досить схожі, але кожна з них наслідує власну бізнес модель та націлена на специфічну категорію користувачів.

З метою усунення недоліків запропонованих систем та створення унікального продукту для користування саме працівниками маркетингу чи сейлз відділу певної айти компаній було прийнято рішення розробити таку системи управління портфоліо, яка буде легкою та зручною в користуванні, допоможе тримати портфоліо в актуальному стані та презентабельному вигляді, надасть можливість автоматизації певних процесів його менеджменту, а також буде в максимальній мірі корисна для залучення нових клієнтів.

РОЗДІЛ 2

ВИБІР ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ

Одним із ключових етапів створення будь-якої комплексної інформаційної системи є аналіз та вибір технологій для створення програмного продукту. Тому спочатку треба проаналізувати існуючі технології та зробити свій вибір на користь тих, що зможуть забезпечити найбільш гнучкий підхід до проектування та розробки програмного забезпечення системи.

На сьогодні існує безмежна кількість різноманітних технологій для створення будь-яких програмних рішень, що можливо реалізувати за допомогою клієнт-серверної моделі взаємодій. Для вибору найсучасніших та найбільш підходящих технологій для розробки потрібно вивчити найпопулярніші технології, методи та підходи їх використання в реаліях інформаційного світу.

2.1. Вибір мови програмування

Почнемо з основного – вибору мови програмування. Переглянувши статистику досліджень (рис. 2.1) з онлайн-ресурсу індексу популярності мов програмування **PYPL** (**P**opularit**Y** of **P**rogramming **L**anguage), побачимо, що лідируючі позиції у сучасному світі займають Python та Java [8].

Worldwide, Oct 2022 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	28.3 %	-1.8 %
2		Java	17.2 %	-0.9 %
3		JavaScript	9.69 %	+0.4 %
4		C#	7.2 %	-0.2 %
5		C/C++	6.45 %	-0.5 %
6		PHP	5.39 %	-0.9 %
7		R	4.03 %	+0.3 %

Рис. 2.1 – Актуальний рейтинг мов програмування

Було вирішено зробити огляд саме цих двох мов програмування, розглянути їх особливості та порівняти ключові характеристики.

Високу популярність мова Python здобула відносно недавно (рис. 2.2), але вже за цей короткий час повною мірою показала свою цінність в багатьох сферах розробки програмного забезпечення. Така поширеність значною мірою зумовлена простотою вивчення, а також дуже швидким ростом спільноти розробників, які щодня впроваджують нові вдосконалення.

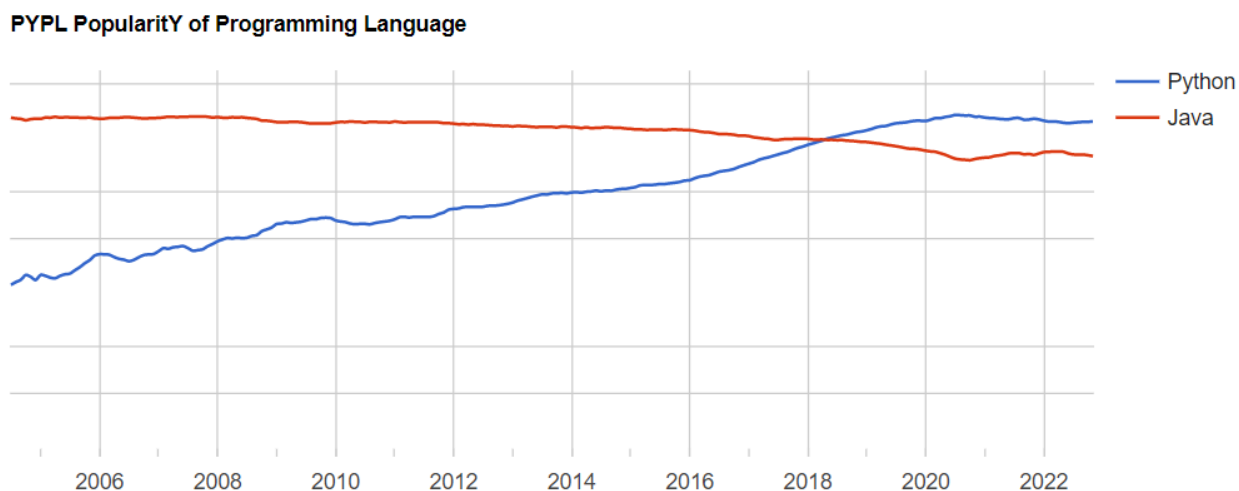


Рис. 2.2 – Зміна лідируючої позиції з рейтингу мов програмування

Python – це високорівнева мова програмування, що орієнтована на підвищення продуктивності розробника та значне полегшення читання коду. Python належить до інтерпретованих мов програмування, що є дуже корисним у процесі налагодження програми, адже така особливість передбачує виконання коду рядок за рядком. Стандартний інтерпретатор реалізований на мові C, а його вихідний код доступний будь-кому у відкритому доступі. Хоча завжди існує можливість розширення існуючої реалізації для отримання нової з власними доповненнями [9].

Характерною ознакою Python є динамічна типізація, тобто розробник може заздалегідь не вказувати тип даних. Синтаксис мови – дуже прозорий, чіткий та стислий, що дає можливість легко освоїтись та досить швидко почати писати код, а також це сильно спрощує читання чужого коду. Розробнику надається

можливість не стежити ретельно за використанням пам'яті програмою, адже інтерпретатор автоматично керує пам'яттю. За допомогою цієї мови програмування можна досить легко працювати зі складними структурами даних та динамічним зв'язуванням, що пришвидшує розробку програми [10].

Python надає можливість мультипарадигменного програмування. Основна увага приділяється функціональній та об'єктно-орієнтованій парадигмі, але підтримуються також процедурна, імперативна та інші [11]. Переважна більшість інтерпретованих мов є кросплатформними. Перенесення програмного забезпечення, написаного на Python, з однієї платформи на іншу не потребує додаткових зусиль, але попередньо потрібно завантажити та скомпілювати відкритий код інтерпретатора.

Суттєвою особливістю, яка свідчить на користь Python, є наявність безмежно широкого спектру бібліотек. Серед такого великого асортименту інструментів завжди знайдеться щось корисне для будь-якої галузі інформаційних технологій, що робить цю мову гнучкою при розробці різноманітних типів проектів. З коробки стандартна бібліотека Python включає безліч компонентів для роботи з мережевими протоколами, засоби для маніпуляції та представлення різних даних, елементи парсингу поштових повідомлень, готові модулі для створення та налаштування HTTP-сервера, плагіни для взаємодії з популярними форматами JSON, CSV, XML тощо. Доступні пакети для роботи з ресурсами операційної системи, unit-тестами, мультимедійними форматами, регулярними виразами, серіалізацією та десеріалізацією об'єктів, криптографічними протоколами, форматуванням текстових представлень тощо.

Окрім стандартної, у вільному доступі для всіх існує необ'ємна кількість гарно описаних прикладних бібліотек: тут є веб-розробка, робота з базами даних, обробка зображень, створення систем штучного інтелекту, високопродуктивне застосування чисельних методів, створення ігор та безліч іншого [12].

Хоча код мовою Python інтуїтивно зрозумілий та легко сприймається, головним мінусом є відносно невисока швидкість виконання програм, оскільки

мова є інтерпретованою. Але Python набуває переваг у моменти, коли критична не швидкість виконання вихідного коду, а швидкість та простота його написання для створення MVP (minimum viable product). Як правило, реалізація на Python у декілька разів коротша, аніж аналогічна програма мовою Java Чи C++. Під час першого запуску модуля інтерпретатор Python генерує .рус файл зі скомпільованим байт-кодом, щоб не витратити час повторно на наступну компіляцію.

Пайтон-програма може використовувати коди, скомпільовані мовами C чи C++, що теж розширює сфери використання. Python має власне доволі зручне API для впровадження модулів на C чи C++, оскільки в деяких задачах доцільніше перекласти вагому частину високотривалих обчислень на іншу, більш швидку мову програмування.

Як вже було зазначено, Python характеризується динамічною типізацією (типи змінних визначаються на етапі безпосереднього виконання програми), що є значущим фактором для збільшення швидкості розробки застосунків. Також за допомогою вбудованих методів Python з коробки надає можливість працювати з цілочисельними значеннями будь-якої довжини, байтами комплексними числами, рядками символів *.unicode* тощо. За замовчуванням мова підтримує операції взаємодії з такими структурами даних як кортежі, списки, множини, асоціативні масиви, байтові послідовності. Додатково, стандартна бібліотека *collections* надає можливості оперувати складнішими структурами даних, як-от: іменовані кортежі, черги, зв'язані списки, стек та ін.

Як показує дослідження від JetBrains (рисунк 2.3), Python на сьогодні найчастіше використовують у розробці веб-застосунків [13].

What do you use Python for?

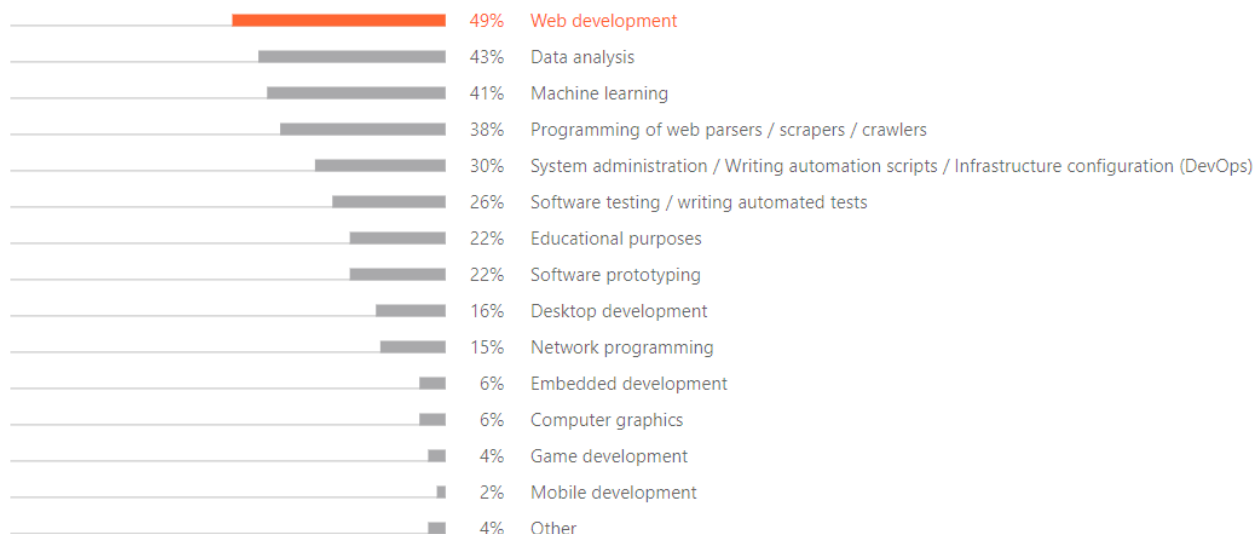


Рис. 2.3 – Сфери застосування Python

Java – це обчислювальна платформа, а також мова програмування, яку створила компанія Sun Microsystems. Від скромних початків вона еволюціонувала до значущої частини сучасного цифрового світу, надавши дуже надійну платформу, яку сьогодні використовують багато служб та програм.

Java – широко використовувана мова програмування, це об’єктно-орієнтована мова, яку саму по собі можна використовувати у якості платформи [14]. Java позиціонується як швидка, надійна та безпечна мова для кодування будь-яких задач: від простих мобільних застосунків і корпоративного програмного забезпечення до високонагружених сервісів і систем для обробки великої кількості даних.

Загальновідомою характеристикою додатків платформи Java є вміння працювати на мільярдах різноманітних пристроїв, включаючи ноутбуки, ігрові консолі, мобільні пристрої, медичне обладнання та багато іншого.

Платформа Java – це набір програмних засобів, що дозволяють програмісту ефективно розробляти й впроваджувати програмні рішення. Сюди входять компілятор, механізм для виконання та стандартний набір бібліотек [15].

Правила та синтаксис мови запозичені з C та C++. За основу була взята об'єктна модель мови C++ та потім модифікована для уникнення конфліктних ситуацій, які зазвичай виникали через помилки розробника, що значно полегшило процес створення об'єктно-орієнтованих програм. Також ряд задач, що в C++ мав виконувати власне програміст, в Java перекладено на віртуальну машину. Віртуальна машина Java (JVM) – це специфікація, яка забезпечує середовище виконання, у якому можна запускати байт-код Java. Це означає, що JVM створює платформу для запуску байт-коду і перетворення його на різні низькорівневі машинні мови, які апаратне забезпечення комп'ютерів може зрозуміти.

Майже для будь-якої мови програмування потрібне спеціальне середовище, що складається з усіх необхідних компонентів, інтерфейсів прикладного програмування та бібліотек для розробки, компіляції, налагодження та виконання програм. Java має два таких середовища: JRE, JDK (рисунок 2.4).

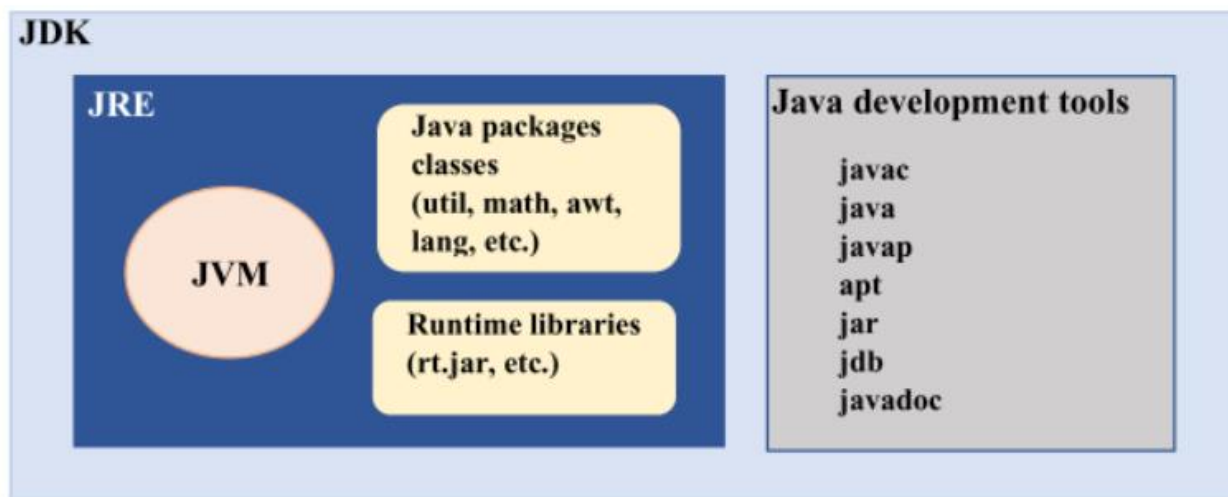


Рис. 2.4 – Архітектура Java

Java Runtime Environment (JRE) забезпечує мінімальні потреби для виконання програми. Середовище складається з віртуальної машини JVM, пакетів ядра Java, класів та допоміжних файлів.

Java Development Kit (JDK) – це середовище розробки програмного забезпечення, яке використовується для створення програм і аплетів Java. Воно

включає середовище виконання Java (JRE), інтерпретатор/завантажувач (Java), компілятор (javac), архіватор (jar), генератор документації (Javadoc) та інші інструменти, необхідні для розробки [16].

Основна перевага Java полягає в тому, що її розроблено для можливості роботи на різноманітних платформах із концепцією **WORA** — «*напиши один раз, запусти будь-де*». Хоча такі мови, як C++, компілюють свій вихідний код, щоб відповідати лише певній платформі та запускатися нативно на її ОС і апаратному забезпеченні, вихідні коди Java компілюються в проміжний байт-код (файл .class) за допомогою компілятора *javac*, що постачається вбудованим у JDK. Цей байт-код у шістнадцятковому форматі з рядками коду операції-операнда, і JVM може інтерпретувати ці інструкції (без подальшої повторної компіляції) на рідну машинну мову, яку може зрозуміти операційна система і основна апаратна платформа. Таким чином байт-код діє як незалежний від платформи проміжний стан, який можна переносити між будь-якими JVM, незалежно від основної ОС та архітектури обладнання. Однак, оскільки віртуальну машину Java розроблено для роботи та зв'язку з основним обладнанням і структурою ОС, спочатку потрібно вибрати відповідну версію JVM для обраної версії ОС (Windows, Linux, Mac) і архітектури процесора (x86, x64) [17].

Оскільки Java є незалежною від платформи мовою, то вона має набагато менше низькорівневих можливостей взаємодії з апаратним забезпеченням, з чого слідує, що така програма виконується повільніше, ніж аналогічне рішення, наприклад, мовою C++. Але для задоволення таких потреб Java надає можливість виклику підпрограм, створених за допомогою інших мов.

Java притаманна статична типізація (типи створюваних об'єктів визначаються на етапі компіляції та не можуть бути змінені в процесі безпосереднього виконання програми), що збільшує час розробки, але надає можливість уникнути багатьох помилок ще на стадії компіляції програми, а не в рантаймі.

Віртуальна машина автоматично керує пам'яттю. Java має вбудований збирач сміття (Garbage Collector), що звільняє розробника від пильного керування пам'яттю та значно пришвидшує процес розробки ПЗ.

Стандартна бібліотека класів Java — це набір динамічно завантажуваних бібліотек, які мова віртуальної машини Java (JVM) може викликати під час виконання. Оскільки платформа Java не залежить від конкретної операційної системи, програми не можуть покладатися на будь-яку бібліотеку. Натомість, платформа Java надає повний набір стандартних бібліотек класів, що містять функції, загальні для сучасних операційних систем: вони надають програмісту добре відомий набір корисних засобів, таких як класи контейнерів і обробка регулярних виразів. Також бібліотека надає абстрактний інтерфейс для задач, які зазвичай напряму залежать від апаратного забезпечення та операційної системи, таких як доступ до мережі чи маніпуляція файлами. Деякі базові платформи можуть не підтримувати всі функції, які очікує програма Java. У цих випадках реалізація бібліотеки може або емулювати ці функції, або забезпечувати послідовний спосіб перевірки наявності конкретної функції.

What types of software do you develop with Java?

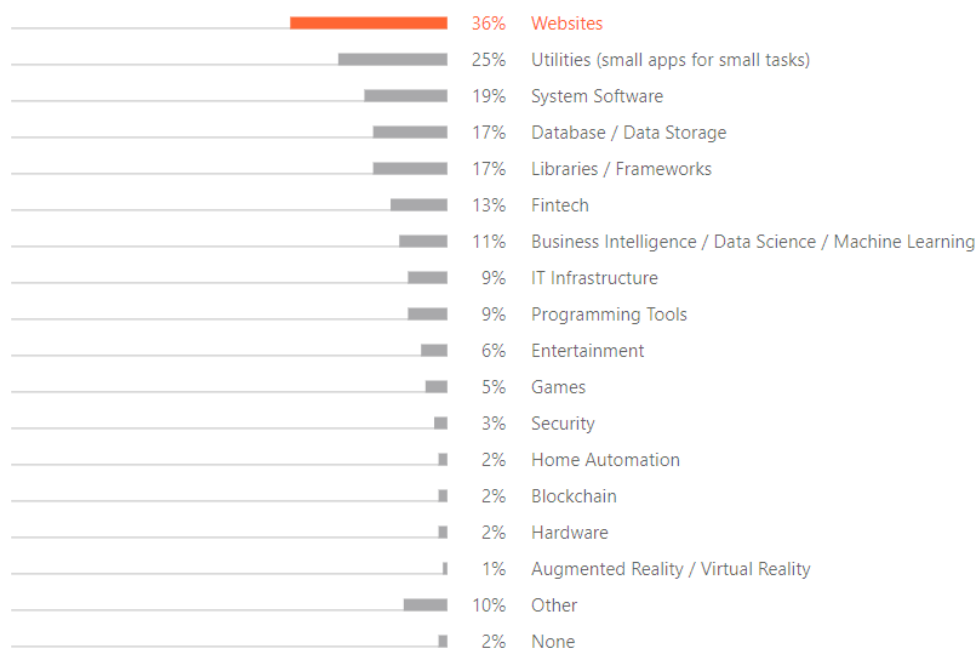


Рис. 2.5 – Сфери використання Java

З досліджень використання Java від JetBrains (рисунок 2.5), можна побачити, що найширшого застосування мова здобула у сфері розробки веб-додатків [18].

Таблиця 2.1

Порівняльна характеристика Python та Java

Особливість	Python	Java
Популярність	Більш популярна	Досить популярна
Мова	Інтерпретована	Компільована
Типізація	Динамічна	Статична
Синтаксис	Легко запам'ятовується	Набагато складніший
Сфери використання	Веб застосування, аналітика даних, машинне навчання, скрапери, парсери, обробка зображень, скрипти	Веб застосування, утиліти, системне ПЗ, мобільні додатки, корпоративні програми, вбудовані системи.
Кількість коду	Менше коду	Значно більше коду
Продуктивність	Повільніша, оскільки визначає тип даних під час виконання програми	Швидше виконання програми за рахунок попередньої компіляції
Швидкість розробки	Швидка розробка завдяки легкості, простоті та практичності	Проекти, як правило, займають набагато більше часу та апаратних ресурсів
Вивчення	Легше у вивченні, синтаксис досить простий і зрозумілий	Навчання займає більше часу, високий поріг входження

Виконавши порівняльний аналіз (таблиця 2.1) основних характеристик Python та Java, було надано перевагу мові програмування Python, як більш гнучкій та зручній під час розробки програмного рішення.

2.2. Вибір фреймворку

У сучасному світі неможливо уявити розробки будь-якої складної системи без використання спеціалізованих фреймворків тієї чи іншої мови програмування. Фреймворк – програмна платформа, яка визначає цільову структуру системи, а також значно полегшує розробку програмного забезпечення, впроваджуючи стандартні загальновідомі практики об’єднання багатьох компонентів у єдину систему.

Звернувшись до цьогорічної статистики JetBrains (рисунок 2.6), було зроблено вибір на користь фреймворку Django, що займає лідируючу позицію серед Python розробників для створення веб-застосунків [13].

What web frameworks or libraries do you use in addition to Python?

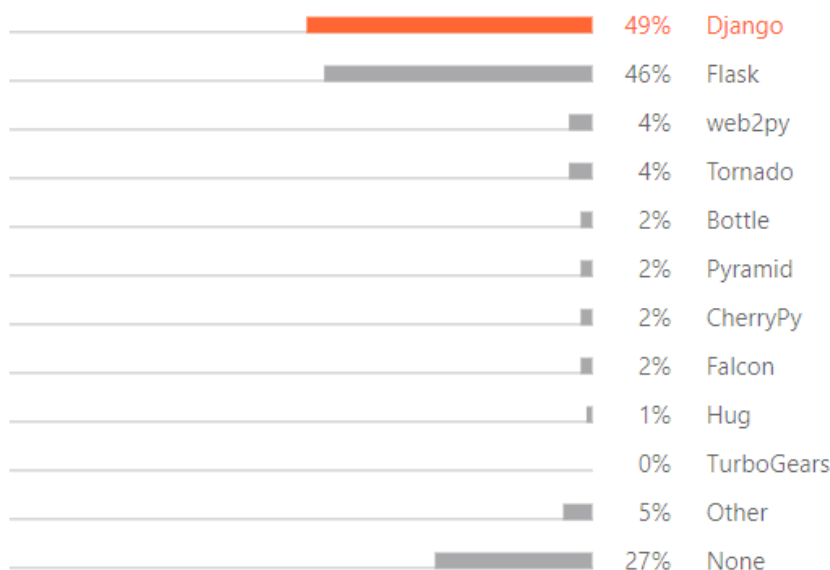


Рис. 2.6 – Рейтинг Python-фреймворків та бібліотек для веб-розробки

Django – високорівневий фреймворк для розробки веб-додатків на Python, що надає змогу швидко та якісно створювати безпечні застосунки, які легко підтримувати в подальшому. Платформа була створена досвідченими фахівцями та бере на себе значну частину задач веб-розробки, що дозволяє повністю зосередитись на створенні веб-додатків без необхідності «вигадувати велосипед». Фреймворк має чітко описану документацію, відкритий код

розповсюджується безкоштовно, а також часто удосконалюється завдяки великому співтовариству розробників-користувачів [19].

Завдяки Django можна швидко та надійно створювати повноцінні програмні рішення за принципом «усе в комплекті», оскільки значну частину засобів розробки та потрібних інструментів платформа надає «з коробки» одразу після встановлення. Усе, що потрібно для веб-застосувань, постачається й працює разом, відповідає найсучаснішим стандартам проектування складних систем, а також супроводжується актуальною документацією.

На фоні інших фреймворк відзначається своєю різноплановістю застосування. Таким чином Django може використовуватись для реалізації бізнес-задачі будь-якої складності: чи то система управління контентом, чи сучасний сайт новин, чи навіть високомасштабована соціальна мережа.

Багатофункціональність проявляється у можливості взаємодіяти з будь-яким клієнтським середовищем чи сучасним форматом відтворення даних, як-от: HTML, XML, JSON тощо).

Знаною перевагою платформи є дуже потужний функціонал аутентифікації, авторизації й адміністрування застосунку, що постачається з коробки та відносно легко налаштовується. Таке рішення задовольняє потреби більшості клієнтів. Хоча фреймворк і надає широкий спектр можливостей за замовчуванням, але за потреби завжди є можливість розширити цей функціонал за допомогою існуючих сторонніх компонентів та бібліотек або ж власних зручних доповнень.

Django спроектований так, щоб допомогти розробнику зменшити велику кількість помилок безпеки. Фреймворк має вбудовану автоматичну систему захисту даних. Наприклад, платформа дозволяє безпечно управляти обліковими записами та паролями створених користувачів, уникаючи при цьому розповсюджених помилок, як-от: зберігання «сирого» пароля клієнта у базі даних (у Django натомість зберігається лише хеш цього пароля) або ж інформації про сесію користувача у файлах cookie (Django поміщає лише ключ у файли cookie, а всі дані зберігаються безпосередньо в БД).

Фреймворк використовує криптографічне хешування паролів та зберігає в базі лише значення хеш-паролів фіксованої довжини. При перевірці введеного паролю Django обробляє його вбудованою хеш-функцією, після чого порівнює результат із збереженим раніше значенням в БД. Таким чином, навіть якщо зломисник отримає збережені хеш-паролі, то визначити справжні значення паролів йому навряд чи вдасться.

Також Django містить особливий набір засобів для захисту від загроз SQL-ін'єкцій, підробки міжсайтових запитів (CSRF), клікджекінгу та міжсайсового скриптингу (XSS).

Слід зазначити, що проекти, створені за допомогою цього фреймворку досить легко масштабуються. Кожен з елементів платформи не залежить від інших, завдяки чому може бути легко змінений чи й зовсім замінений на інший. Таке чітке розділення компонентів свідчить про простоту масштабування з ростом вхідного трафіку. Така взаємодія може відбуватись шляхом розширення будь-якого з рівнів обладнання: сервери додатків, баз даних, кешування, реплікації тощо. Так, наприклад, усесвітньовідома мережа Instagram була успішно масштабована з початкового Django-проекту.

Оскільки платформа була створена за допомогою мови програмування Python, то вона може працювати на переважній більшості існуючих комп'ютерних систем, що надає можливість не зосереджуватись на якомусь конкретному серверному програмному забезпеченні, а запускати Django-проекти на будь-яких версіях та дистрибутивах Linux, Windows чи Mac OS. Також фреймворк користується популярністю серед сучасних хостингових сервісів, які у свою чергу надають потрібну інфраструктуру, а також покрокову інструкцію з розгортання власних застосунків у мережі Інтернет.

Код Django створений, дотримуючись усіх відомих принципів та шаблонів проектування програмного забезпечення, що спрямовані на подальшу підтримку, розширення й повторне використання кодової бази. Головним принципом Django є «*Don't repeat Yourself (DRY)*», що передбачає неможливість повторів коду, а отже й зменшує кількість вихідного коду. На додаток,

фреймворк дозволяє стиснути певні функціональні можливості в окреме єдине рішення (Django application), які потім можна перевикористати без змін у будь-якому іншому програмному рішенні.

Django початково містить власний веб-сервер для тестування та налагодження в процесі розробки застосунку. Він перевіряє зміни в коді програми у режимі реального часу, що робить його дуже зручним інструментом. Фреймворк офіційно підтримує можливість взаємодії з найпопулярнішими базами даних MySQL, PostgreSQL, SQLite, Oracle, MariaDB. Також існують і сторонні бібліотеки для маніпуляції з іншими БД.

Архітектура Django ґрунтується на основі шаблону MVC (Model-View-Controller), але вносить певні корективи та особливості реалізації. «Контролер» (*controller*) з MVC шаблону Django позначає як «вид» (*view*), а те, що називалось «видом» в MVC, в рамках Django-проекту перетворюється на «шаблон» (*template*). Через ці особливості архітектуру назвали MTV (Model-Template-View). На рисунку 2.7 показана взаємодія цих елементів.

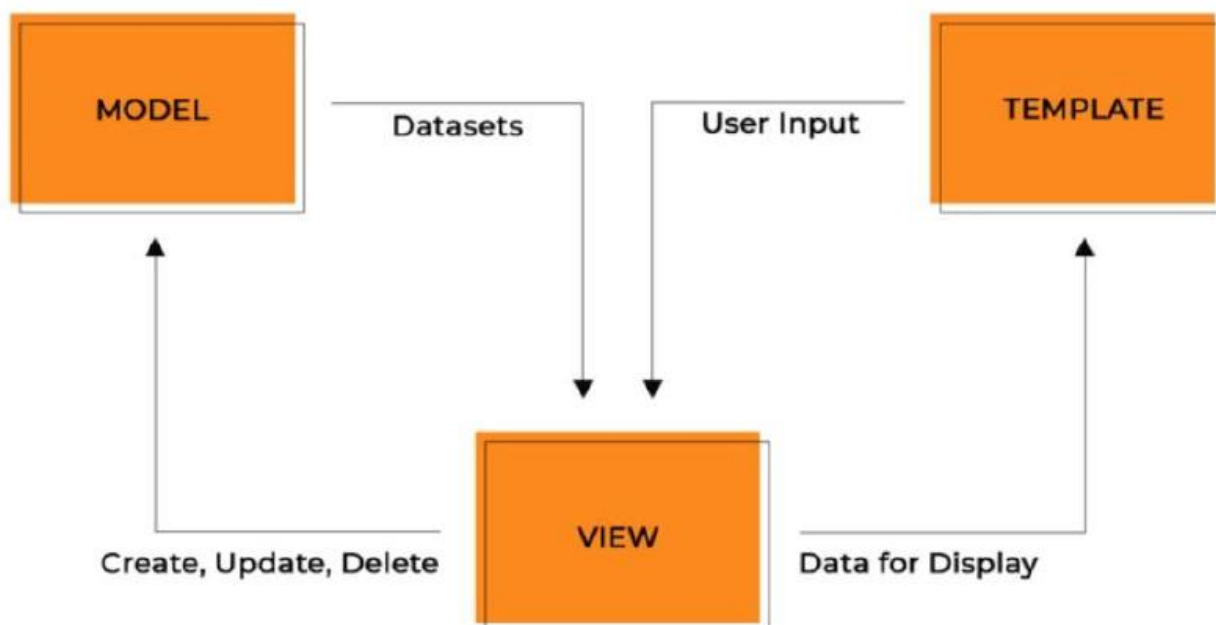


Рис. 2.7 – Архітектура Django-проекту

Фреймворк володіє власним шаблонізатором та впроваджує особисту мову розмітки сторінки з багатьма цікавими інструментами. Шаблони проекту

Django – це файли, що містять текст HTML для візуального відображення інформації. Такі файли можуть включати як статичний, так і динамічний код. Шаблони лише представляють інформацію кінцевим користувачам, а тому не містять в собі ніякої бізнес-логіки. На рисунку 2.8 представлена структура типового шаблону Django.

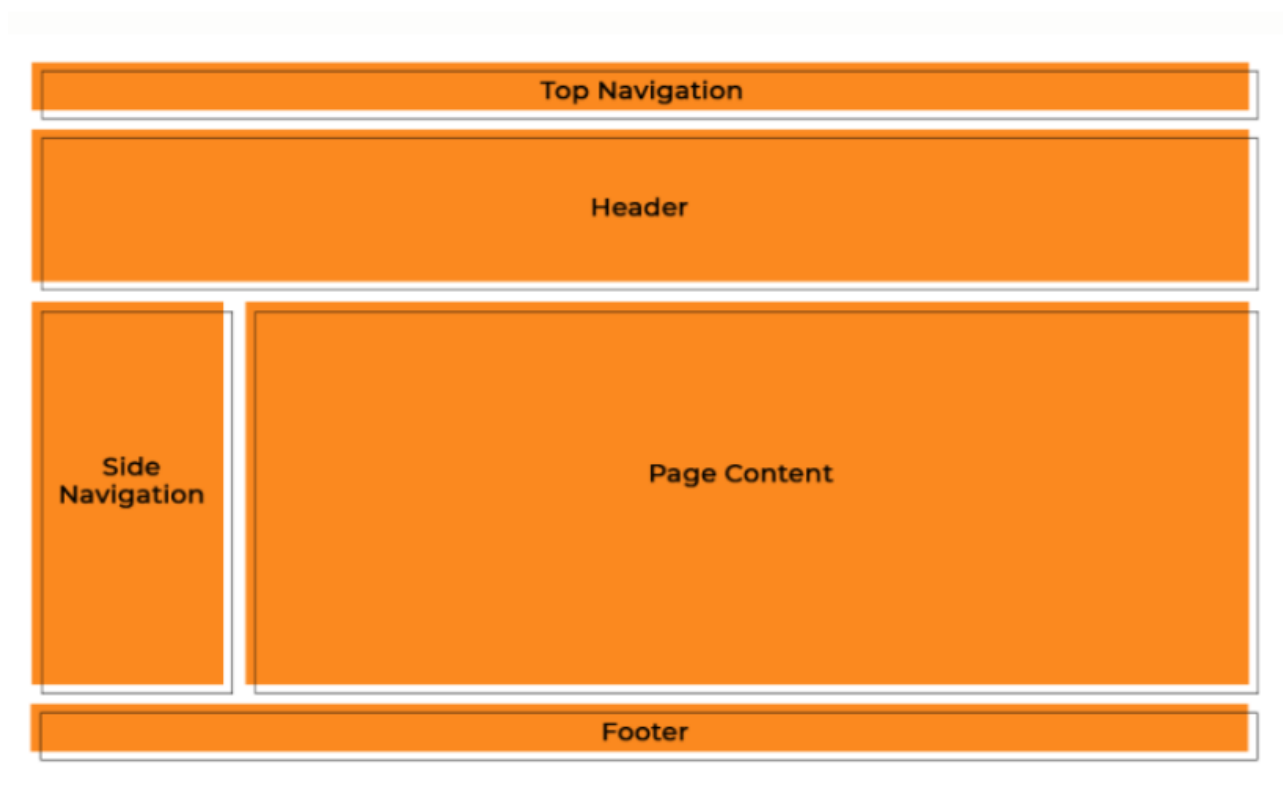


Рис. 2.8 – Структура шаблону Django-проекту

Структурна схема базового проекту на Django

Традиційно, веб-застосунок очікує від клієнтського браузера HTTP запит отримання певних даних або ж виконання дій, що вимагається. Коли такий запит від користувача отримано сервером, застосунок виконує усі необхідні операції на основі заданої URL-адреси та методу запиту. Далі, від того, що вимагається, він може зчитувати, оновлювати дані, виконувати сторонні обчислення чи будь-які інші задачі, необхідні для задоволення потреб клієнта. Після чого додаток налаштовує шаблон відображенні інформації та повертає відповідь браузеру у вигляді HTML-сторінки. Обробка запиту користувача показана на рисунку 2.9.

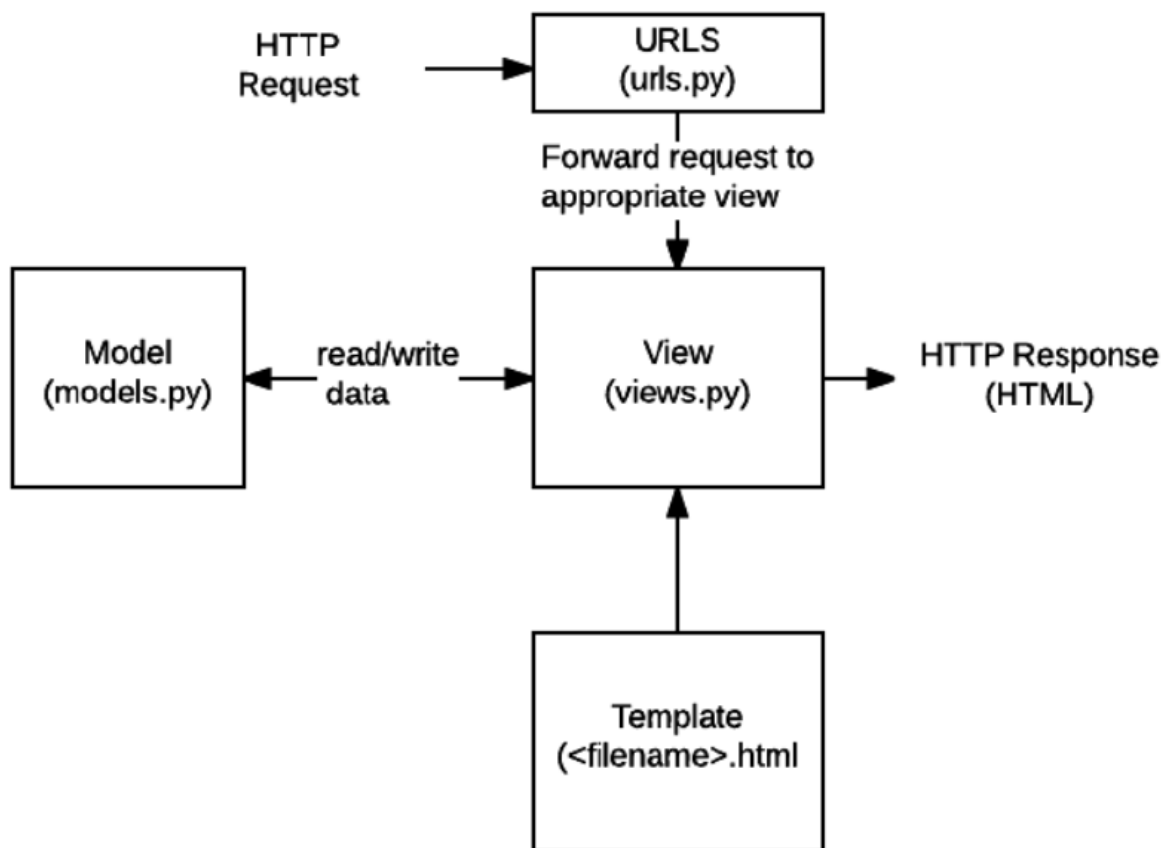


Рис. 2.9 – Структурна схема базового Django-проекту

Зазвичай, логіку обробки кожного такого запиту розміщують в окремому спеціалізованому файлі. Кожен додаток (Django application) має містити принаймні такі основні компоненти: *urls*, *models*, *views*, *templates*.

Urls

Звісно, можна обробляти запити на кожну URL-адресу використовуючи лише одну функцію, але такий підхід значно ускладнює читабельність та розуміння програмного коду, тому прийнято створювати окремі функції для обробки кожної URL окремо. В Django існує URL-диспетчер, що перенаправляє кожен клієнтський запит у відповідне представлення *view* на основі URL-адреси, передаючи усі необхідні позиційні чи іменовані аргументи, що заздалегідь були вказані користувачем в URL-адресі.

Models

Моделі – це такі Python-об’єкти, що повною мірою описують структуру даних веб-застосунку. Кожна окрема модель характеризує відповідну таблицю в

базі даних. Використовуючи особливості об'єктно-орієнтованого підходу мови програмування, моделі дають змогу легко взаємодіяти з БД за допомогою високорівневих структур Python.

Views

View – це функція чи клас, що отримує HTTP-запит, потім обробляє його (виконує вказані інструкції) і повертає відповідь у попередньо визначеному форматі даних (HTML, XML, JSON тощо). Views мають прямий доступ до моделей (тобто до бази даних), завдяки чому повністю можуть задовольнити будь-який запит.

Templates

Шаблон – це такий текстовий файл, що містить спеціалізовану мову розмітки, мета якого відображення структури даних для кінцевого клієнта. Найчастіше, це HTML-файл, що містить усі необхідні поля, що за потреби можуть динамічно розширюватись вмістом за допомогою «видів» (*views*).

Огляд Django ORM

Ключовим фактором, що свідчить на перевагу фреймворку Django є наявність власного ORM, що дозволяє одразу без довгих налаштувань приступити до роботи з найпопулярнішими базами даних.

Object-Relational-Mapping (ORM) – це об'єктно-реляційне відображення, тобто прикладний програмний інтерфейс, що дозволяє пов'язати концепції баз даних із об'єктно-орієнтованою парадигмою програмування. Це надає можливість доступу до БД та маніпуляції даними, використовуючи засоби ООП. Таким чином, таблиці БД являють собою класи Python, поля представляються у вигляді атрибутів цих класів, а операції взаємодії реалізуються через визначення методів класів.

Користуючись ORM, слід пам'ятати, що цей підхід потребує більше часу для виконання запиту до БД, оскільки фреймворк спочатку повинен транслювати об'єктно-орієнтований код в низькорівневу транзакцію, а тільки потім виконати її. Проте з іншою сторони, такий інтерфейс взаємодії значно пришвидшує розробку програмних застосунків, а також не потребує володіння мовою для

керування вмістом бази даних. Більш того, така реалізація є більш гнучкою, бо дає змогу за потреби майже без змін коду дуже легко змінювати реляційне сховище даних.

2.2. Вибір сховища даних

Система управління базами даних (СУБД) – це набір програмних та мовних засобів, що призначені для налаштування та керування вмістом баз даних (взаємодія може виконуватись з боку декількох користувачів одночасно).

Реляційна база даних – це сукупність даних, пов’язаних між собою певним чином. Такі дані зазвичай представлені у вигляді таблиць: кожен стовбець позначає деякий тип даних, а кожна комірка таблиці зберігає певне значення атрибута. Кожен окремий рядок визначає набір пов’язаних полів, що повною мірою характеризують той чи інший об’єкт у конкретній системі. Як правило, кожен запис (рядок) позначається первинним ключем (унікальним ідентифікатором), а записи кількох різних таблиць можуть зв’язуватись між собою за допомогою зовнішнього ключа.

Нереляційна база даних – це така БД, в якій використовується не табличне представлення даних, а, наприклад, документне, графне чи будь-яке інше, що гарно оптимізоване спеціально під конкретні вимоги типів даних для збереження.

Основним сховищем даних було вирішено обрати реляційну БД, оскільки для реалізації проектів такого типу вона підійде краще. Цьогорічне опитування розробників на порталі *Stack Overflow* (рисунок 2.10) показало, що лідируючі позиції в рейтингу сучасних СУБД займають **MySQL** та **PostgreSQL**. Отож порівняємо їх особливості для вибору більш підходящого інструменту.

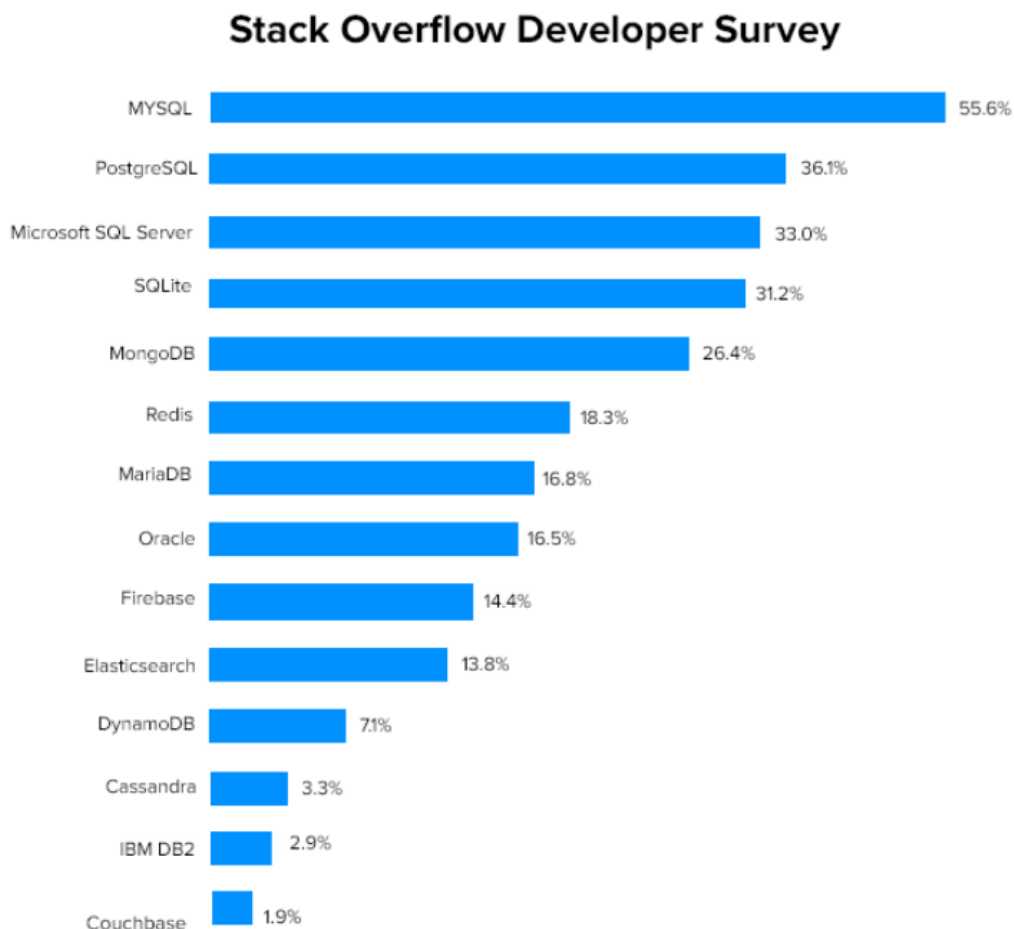


Рис. 2.10 – Рейтинг популярності СУБД серед розробників усього світу

Збереження даних

MySQL може використовувати різні двигуни для збереження даних в таблицях, але взаємодія з цими двигунами прихована глибоко в системі управління. СУБД підтримує більшість сучасних двигунів (*InnoDB*, *MyISAM*, *BerkeleyDB*), а різниця між ними полягає лише в методах запису та читання даних з дискового простору того чи іншого обладнання. Також слід відзначити, що ці інструменти ніяк не впливають на синтаксис побудови структурних запитів до БД, а також на їх безпосереднє виконання.

PostgreSQL позиціонується як об'єктно-реляційна система управління базами даних, що в свою чергу забезпечує можливість представлення усіх таблиць у вигляді об'єктів, дозволяє виконувати наслідування та взаємодіяти з даними використовуючи об'єктно-орієнтовані функції. За замовчуванням, двигун для читання та збереження – *Storage Engine*.

Відповідність стандартам SQL

MySQL не має змоги підтримки будь-яких новітніх можливостей структурованої мови запитів. Цей вибір розробники технології зробили задля спрощення використання. СУБД намагається відповідати більшості стандартам, але іноді відмовляється від цього на користь простоті застосування. Також часто для покращення зручності використання *MySQL* розробники додають окремі розширення незважаючи на стандарт.

PostgreSQL – це проект, що пропонує повністю відкритий вихідний код. Згідно з офіційною документацією, система покриває 160 із 179 критеріїв, що задовільняють структурну мову запитів [20]. Розробники зосередились на стандартах SQL та щоразу втілюють усі найновіші функції та можливості. А з іншої сторони, це сильно ускладнює систему. Мабуть, саме це спричиняє меншу популярність *PostgreSQL*.

Обробка даних

PostgreSQL має значно більше можливостей обробки даних, оскільки містить більше функцій. СУБД надає можливість створювати індекси одразу по декільком полям таблиці. Крім того, підтримуються індекси різних типів, як-от: b-tree, r-tree, hash, GIN, GIST тощо; а також розробнику надається можливість створювати власні типи індексів [21]. На додаток, система підтримує рекурсивні запити, наслідування таблиць, регулярні вирази та ін.

Можливості застосування

Існує декілька реалізацій *MySQL*: програмне забезпечення з безкоштовною ліцензією та відкритим кодом для спільноти (не містить повного функціоналу) та сукупність платних версій продукту для комерційного використання (з усім необхідним набором функцій).

Оскільки вихідний код *PostgreSQL* є повністю відкритим для будь-кого, то дана СУБД забезпечує відкриту ліцензію для усіх видів розробки, а також супроводжується чіткою офіційною документацією [22].

Надійність використання

Безпека системи управління базами даних досягається впровадженням колективних засобів захисту від шкідливого незаконного використання або ж сторонніх загроз чи атак зловмисників.

MySQL веде спеціальні списки управління (Access Control Lists), тобто керує доступом до усіх можливих операцій, що зазвичай вимагає користувач, наприклад, підключення до бази даних, отримання чи оновлення інформацій в таблицях і т.п. Також СУБД додатково шифрує канали передачі між сервером та клієнтом.

PostgreSQL вводить особливу ієрархічну систему ролей для наслідування та керування правами доступу різних користувачів. Окрім стандартного SSL-шифрування з'єднання, існує захист записів на різні таблиці. Також у початковій комплектації СУБД наявний такий інструмент як *SELinux*, що збільшує можливості контролю прав на основі політики безпеки *SELinux*.

Продуктивність СУБД

MySQL працює швидше, оскільки розроблялась для максимальної продуктивності. У свою чергу *PostgreSQL* орієнтована на максимальне наближення до стандартів SQL та надання широкої кількості можливих налаштувань та плагінів.

У переважній більшості випадків *MySQL* використовують з InnoDB. Таблиця, створена за допомогою цього двигуна, має вигляд збалансованого дерева (b-tree) з індексами, що надає можливість дуже швидко зчитувати дані з диску, використовуючи мінімальний набір операцій при цьому.

Основною з областей, де *MySQL* виграє у швидкості, є операції з інтенсивним читанням. Оскільки багато програм лише читають і відображають інформацію з бази даних. Існує багато випадків, коли реалізація з *MySQL* значно перевершувала реалізацію з *PostgreSQL* [23]. Це пояснюється тим, що *PostgreSQL* виділяє більший обсяг пам'яті, коли створює новий процес для кожного з'єднання, що призводить до надмірного використання пам'яті та фактично з'їдає швидкість. Таким чином, СУБД жертвує швидкістю заради цілісності даних і відповідності стандартам.

Проте швидкість є орієнтиром, який визначатиметься на основі того, як використовується база даних. *PostgreSQL* працює швидше, коли має справу з масивними наборами даних, складними запитами та операціями читання-запису. З іншого боку, відомо, що *MySQL* є швидкою лише для команд зчитування, оскільки не може повноцінно підтримувати операції читання та запису одночасно.

Підтримка СУБД

Компанія Oracle, що офіційно володіє та розробляє технологію *MySQL*, має широку аудиторію клієнтів, а тому наряду зацікавлена в підтримці продукту. Хоча зазвичай йдеться про покращення існуючих функцій СУБД, аніж розробку нових.

PostgreSQL характеризується великим співтовариством користувачів, завдяки яким розробники повсякденно удосконалюють існуючі особливості системи, в той же час слідує передовим трендам та впроваджують новий функціонал і політику безпеки.

Порівнявши дві найпопулярніші на сьогодні СУБД, було вирішено віддати перевагу використанню ***PostgreSQL*** через відповідність стандартам, безкоштовність ліцензії, гнучкість та надійність системи.

2.3. Вибір інструменту для виконання відкладених задач

Розробляючи програму, що часто взаємодіє з клієнтським бразером та інтенсивно використовує дані, можна зіткнутися з такою проблемою, коли довгострокові задачі серйозно сповільнюють роботу застосунку. Сучасні користувачі очікують, що сторінки додатку завантажуватимуться миттєво, але для виконання завдань із великим обсягом даних може знадобитися чимало секунд або навіть хвилин.

Якщо постає задача швидко завантажувати та обробляти усі клієнтські запити нашим додатком без зависань, то необхідно зняти частину роботи з веб-серверу. Для вирішення цієї проблеми часто використовують фонові завдання.

Один із способів зробити це – асинхронна обробка. Поки веб-сервер завантажує наступну сторінку або отримує новий запит іншого користувача, другий сервер виконує необхідні обчислення у фоновому режимі. Також це може бути не окремий сервер, а частина ресурсів (спеціально виділені потоки центрального процесора та пам'ять) того, на якому працює основний код застосунку. Такі сервери, що виконують фонові задачі (або ж розділені ресурси одного сервера), називаються «воркерами» (*workers*). Хоча зазвичай розгорнутий один або декілька основних веб-серверів застосунку, що обробляють усі запити користувачів, *воркерів*, що обробляють довготривалі задачі, може бути безліч. Вони можуть вносити зміни в базу даних, оновлювати інтерфейс користувача через веб-хуки або зворотні виклики, додавати елементи до кешу, обробляти файли, надсилати електронні листи, ставити майбутні завдання в чергу тощо. В той же час головний сервер веб-додатку залишається вільним для відповіді на запити користувачів [24].

Для управління та завантаження цих *воркерів* використовується система передачі повідомлень. Це таких метод синхронної або асинхронної взаємодії, що забезпечує комунікацію між окремими частинами застосунку. Зазвичай завдання передаються *воркерам* через чергу повідомлень, що реалізується за принципом «першим прийшов, першим вийшов» (FIFO), тобто задачі обробляються по мірі надходження та в порядку їх додавання. Коли *воркер* стає доступним, він бере першу завдання з черги та починає обробку. Якщо розгорнуто декілька *воркерів*, кожен обирає доручення по порядку надходження. Черга гарантує, що кожен *воркер* отримує лише одне завдання за раз, і що кожна задача обробляється лише одним *воркером*.

Celery – це розподілена черга задач, що базується на асинхронній передачі повідомлень. Створена на Python, але може використовуватись будь-якою мовою програмування за допомогою методу *webhooks*.

Celery дозволяє програмам Python швидко впроваджувати черги завдань для багатьох *воркерів*. Вона піклується про складну частину отримання завдань і відповідного їх призначення. Основний застосунок, що передав задачу, може

продовжувати вільно працювати, а незабаром звернеться до Celery для перевірки статусу виконання або отримання результатів обчислень.

Отож Celery здатна знизити навантаження та розгрузити основний веб-сервер застосунку, асинхронно виконуючи частину обчислень у вигляді відкладених фонових задач.

2.3. Вибір технологій для реалізації веб-інтерфейсу

Для побудови інтерфейсу користувача системи коротко оглянемо основні сучасні інструменти, без яких не відбувається жодна розробка, а також доступні вбудовані технології вже обраного фреймворку.

HTML – мова розмітки гіпертексту, що є ключовим інструментом при створенні будь-якого веб-рішення. Тут описується структура сторінок та розташування інформації [25].

CSS – технологія, що дозволяє зобразити унікальний зовнішній вигляд HTML-файлу, надає змогу описати представлення різних елементів за допомогою новітніх медіа-засобів [26].

Bootstrap – фреймворк, що охоплює широкий набір інструментів для швидкої розробки користувацьких інтерфейсів [27].

Django Template Language – спеціалізована мова створення шаблонів, яка супроводжується сукупністю різних виразів для полегшення та пришвидшення розробки веб-інтерфейсів. Основною особливістю вважається наявність вбудованих тегів, а також можливість створення власних для оперування змінними, форматування чи фільтрації даних, розширення чи наслідування існуючих шаблонів тощо [28].

JavaScript – мова програмування, що постійно використовується для динамічного наповнення та для опису сценаріїв поведінки веб-сторінок. Також застосовується для впровадження гнучкості та інтерактивної взаємодії з клієнтом [29].

jQuery – бібліотека JavaScript, що надає розробнику можливість зручного управління об'єктами структурного документу, використовуючи селектори CSS. Значно спрощує динамічне керування вмістом завдяки Аjax-запитам[30].

ВИСНОВКИ ДО РОЗДІЛУ 2

Проаналізувавши існуючі програмні рішення, було вирішено покласти в основу розробки клієнт-серверну архітектуру взаємодії. У цьому розділі проведено аналіз та порівняння ключових характеристик найпопулярніших технологій для реалізації застосунку. Вирішено використати мову програмування *Python*, як основну для створення бізнес-логіки застосунку. На додаток, розробка буде проводитись за допомогою потужного фреймворку *Django*.

Як основне сховище даних було обрано *PostgreSQL*. СУБД пропонує повністю безкоштовну ліцензію, надає чітко описану документацію, забезпечує надійний захист даних, постачається з повним набором потрібного функціоналу, та відповідає стандартам мови SQL, що робить її дуже гнучкою та підтверджує правильність обраного рішення.

Також було прийняте рішення розгрузити основну програму застосунку, що обробляє запити клієнтів, та додати спеціалізовану програмну бібліотеку *Celery* для асинхронного виконання відкладених задач з довготривалими обчисленнями.

Створення інтерфейсу користувача буде проводитись з використанням стандартних на сьогодні (але від того не менш дієвих) інструментів та технологій: *HTML*, *CSS*, *Bootstrap*, *JavaScript*, *JQuery*, а також окремої мови шаблонізатора *Django*.

РОЗДІЛ 3

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Для розробки бізнес-логіки системи було вирішено обрати клієнт-серверну архітектуру взаємодію, опираючись на загальновідомий шаблон проектування MVC (Model-View-Controller), а конкретніше, дещо видозмінена його реалізація в рамках обраного фреймворку – MTV (Model-Template-View).

Основна ідея шаблону – розділити дані, обробку бізнес-логіки та зовнішній вигляд застосунку на три окремі компоненти так, щоб зміна одного з них мала незначний або ж взагалі ніякий вплив на інші. Цей підхід удосконалює можливості повторного використання кодової бази в інших прикладних додатках, а також спрощує подальшу підтримку та удосконалення програми (зміна користувацького інтерфейсу ніяк не відобразиться на функціях обробки бізнес-логіки). Концепція MTV передбачає розділення архітектури на три ключові елементи: *модель, шаблон, представлення*.

Модель – об’єктна модель прикладної предметної області, що зберігає усі необхідні дані, а також способи взаємодії з цими даними. Модель отримує запити на читання, створення чи зміну інформації в БД, виконує їх та повертає стан, тобто займається лише обробкою даних та ніяк не взаємодіє безпосередньо з користувачем.

Шаблон застосовується для візуалізації даних кінцевому користувачу. Отримані дані можуть бути представлені будь-яким чином.

Представлення керує більшою частиною обробки даних, логікою програми та обміном повідомленнями. Представлення виконує роль сполучника між системою і клієнтським браузером та відповідає за обробку запитів. Може використовувати *модель* і *шаблон* для виконання поставлених задач та відтворення реакції на потреби користувача.

3.1. Структура розробленого застосунку

Структура створеного застосунку зображена на рисунку 3.1.

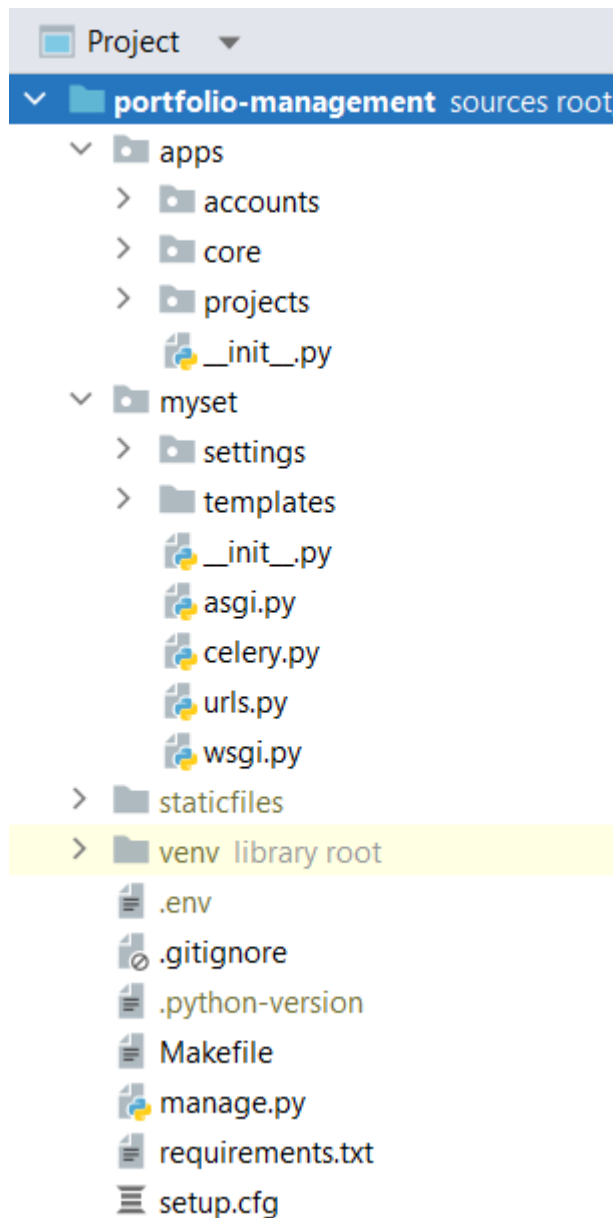


Рис. 3.1 – Структура веб-застосунку

Розроблений проект містить такі компоненти:

- *apps* – директорія для відокремлення логіки створених додатків (Django applications).
- *apps/accounts* – додаток для керування системою аутентифікації, авторизації та профілю користувачів.
- *apps/core* – додаток з базовими утилітами.
- *apps/projects* – додаток для маніпуляції портфолію, управління проектами, поширення обраних наборів робіт.

- *myset* – головний додаток, що управляє застосунком. Є точкою входу та містить усі необхідні налаштування проекту.
- *staticfiles* – директорія, що містить статичні файли (*images*, *.css*, *.js*) для полегшення візуального відображення сервером.
- *venv* – віртуальне оточення Python-інтерпретатора, що було створене для ізоляції залежностей програми від системних бібліотек.
- *.env* – файл зі змінними оточення, що включає секретні ключі, паролі доступу та іншу чутливу інформацію.
- *.gitignore* – елементи проекту, які не варто відстежувати системою контролю версій (як наприклад, секретні змінні оточення).
- *.python-version* – файл, що вказує використану при розробці версію мови програмування для успішної інтеграції необхідних залежностей програми.
- *Makefile* – спосіб автоматизації процедури розгортання застосунку й виконання типових команд проекту.
- *manage.py* – скрипт-утиліта для керування проектом на Django.
- *requirements.txt* – визначений список залежних бібліотек для коректної роботи веб-додатку.
- *setup.cfg* – конфігурація допоміжних інструментів розробки (linter, code formatter, static type checker).

Кожен додаток в директорії *apps* відповідає за окрему частину логіки застосунку та містить ключові елементи підпрограм: послідовність міграцій, набір шаблонів візуалізації, об'єктні моделі опису даних, логіку обробки представлень (*views*), утиліти для реалізації допоміжних функцій та багато іншого.

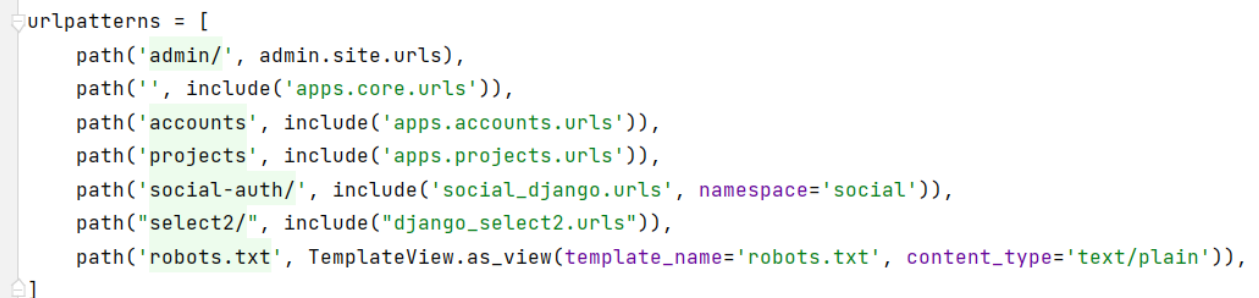
3.2. Побудова веб-сервера за допомогою Django

Розробка серверної частини на Django починається зі створення проекту за допомогою команди *django-admin startproject myset* та попереднього визначення файлу налаштувань, місцезнаходження якого можна вказати одразу у файлі

керування застосунком. За замовчуванням сценарій запуску Django-серверу шукає файл під назвою *settings.py* у внутрішньому каталозі *myset*. Цей файл містить усі види конфігурації для конкретного проекту Django. Як правило, усі вони визначаються у верхньому регістрі, наприклад: `TEMPLATES`, `DATABASES`, `LANGUAGES`, `MIDDLEWARE` тощо.

Маршрутизація системи відбувається завдяки конфігурації URL-адрес. Найважливіший параметр в налаштуваннях називається `ROOT_URLCONF`. Він повідомляє Django, який модуль Python слід використовувати як конфігурацію URL для цього веб-застосунку. Автоматично згенерований файл *settings.py* містить налаштування `ROOT_URLCONF`, що вказує на автоматично згенерований початковою командою файл *urls.py*:

```
ROOT_URLCONF = "myset.urls"
```



```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('apps.core.urls')),
    path('accounts', include('apps.accounts.urls')),
    path('projects', include('apps.projects.urls')),
    path('social-auth/', include('social_django.urls', namespace='social')),
    path("select2/", include("django_select2.urls")),
    path('robots.txt', TemplateView.as_view(template_name='robots.txt', content_type='text/plain')),
]
```

Рис. 3.2 – Головний конфігураційний файл URL-шаблонів

Таке значення відповідає файлу *myset/urls.py*. Коли надходить запит для певної URL-адреси, скажімо, запит */home/*, Django завантажує `URLconf`, на який вказує параметр `ROOT_URLCONF`, а також усі необхідні конфігураційні залежності з інших додатків (такого виду файл продемонстровано на прикладі *myset/urls.py*, що зображено на рисунку 3.2). Потім перевіряє кожен із шаблонів URL-адрес у цій конфігурації по порядку, порівнюючи запитувану URL-адресу з шаблонами по одному, поки не знайде відповідний.

Коли веб-застосунок знаходить відповідність, він викликає функцію обробки (*view*), пов'язану з цим шаблоном, передаючи об'єкт *HttpRequest* як перший параметр. Відповідно, функція представлення повинна повертати об'єкт типу *Response* (це може бути *HttpResponse*, *JsonResponse*, *FileResponse* та інші).

Виконавши ці дії, далі Django зробить все інше, перетворюючи об'єкт Python на правильну веб-відповідь із відповідними заголовками та тілом HTTP протоколу (тобто вмістом веб-сторінки для користувача).

На додаток до простого зіставлення URL-адреси з шаблонами, Django забезпечує досить високу гнучкість обробки запитів. Типовий процес обробки та перетворення `URLconf` на функцію представлення, яка повертає `HttpResponse`, можна скоротити або розширити за допомогою такого проміжного програмного забезпечення як *middleware*.

Middleware – це додаткове ПЗ, що підключається до обробки запитів та відповідей Django. Представлене у вигляді низькорівневої системи «плагінів» для глобальної зміни введення або виведення. Ці обробники запускаються ще перед потраплянням запиту до функції представлення, а також по завершенню її виконання перед відправкою відповіді. Код *middleware* відпрацьовує одразу після отримання запиту, може змінювати його й передавати далі або ж одразу повертати помилку без потреби подальшої обробки методами *view*. Кожен компонент проміжного програмного забезпечення відповідає за виконання певної функції. Наприклад, Django-проект містить таку складову як `AuthenticationMiddleware`, що пов'язує користувачів із запитами за допомогою системи сесій. Також *response middleware* добре підходить для постобробки `HttpResponse` безпосередньо перед тим, як його буде надіслано в браузер або для очищення ресурсів, пов'язаних із запитом. Конфігурація *middleware* та порядок їх запуску визначається у файлі налаштувань проекту.

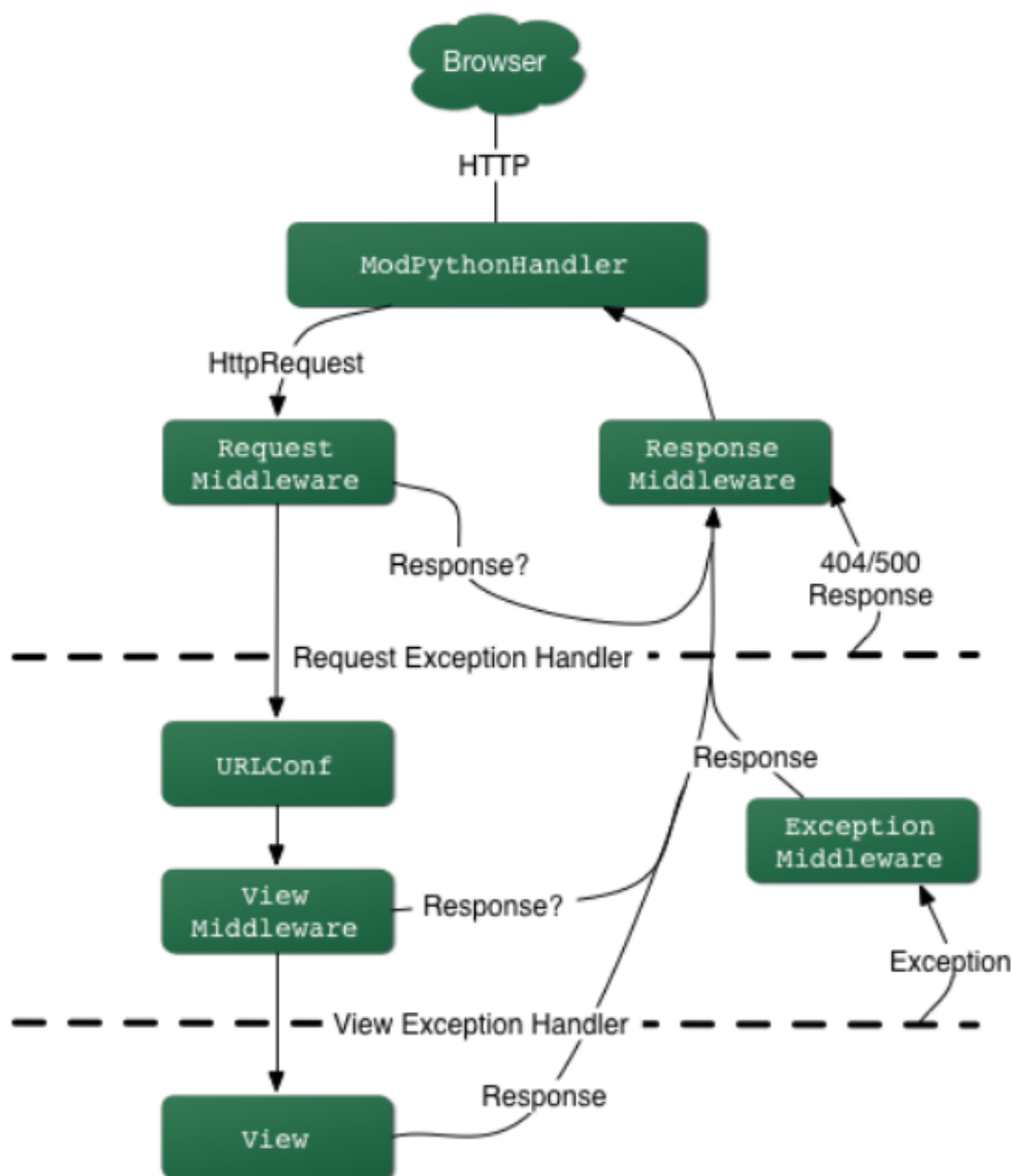


Рис. 3.3 – Алгоритм обробки запиту

Таким чином можна чітко сформулювати повний шлях HTTP-запиту (рисунок 3.3) від клієнтського браузера до створеного Django застосунку та повернення відповіді:

1. Користувач виконує запит на конкретну URL-адресу.
2. Запит проходить через деякий шар проміжного програмного забезпечення *Request Middleware*, яке може змінювати запит або ж одразу створювати відповідь.
3. Застосунок визначає кореневу конфігурацію URL, переглядаючи налаштування *ROOT_URLCONF*. Django порівнює всі URL-шаблони

в `URLconf`, шукаючи перший, який відповідає адресі запиту, після чого знаходить функцію представлення, що пов'язана з URL-адресою.

4. Викликаються проміжні програми *View Middleware*, які можуть маніпулювати запитом або відповідати на нього.
5. Запит передається у функцію представлення, яка після його обробки повертає відповідь.
6. Потім відповідь (*response*) проходить через *Response Middleware*, де може доповнитись у разі необхідності.
7. Django перетворює *response* на належну відповідь HTTP, результатом чого є веб-сторінка.

Вміст проекту працює з трьома основними блоками: URL-адресами, шаблонами та додатками (Django applications), які були створені окремо, але підключаються один до одного для доставки вмісту та обміну повідомленнями, що є частиною слабозв'язаних принципів проектування архітектури Django.

URL-адреси визначають точки входу або місце доступу до контенту. Шаблони визначають кінцеві точки, які надають форму збереженому вмісту даних. А додатки служать проміжним програмним забезпеченням між URL-адресами та шаблонами, додаючи або змінюючи вміст бази даних, та взаємодіючи з користувачем. Різні компоненти розроблялись у залежності від їхнього призначення: для запуску статичного вмісту достатньо було лише створити та налаштувати URL-адреси та шаблони Django, а для додатків з динамічно змінюваним контентом (створеного з бази даних та безпосередньої взаємодії користувача) створювались та налаштовувались спеціалізовані підпрограми Django.

Але перш ніж описувати, як створювати та налаштовувати додатки Django, дуже важливо зрозуміти, як кожна з цих частин працює одна з одною. На рисунку 3.4 показано робочий процес Django для користувацьких запитів і те, як вони працюють з URL-адресами, шаблонами та підпрограмами Django.

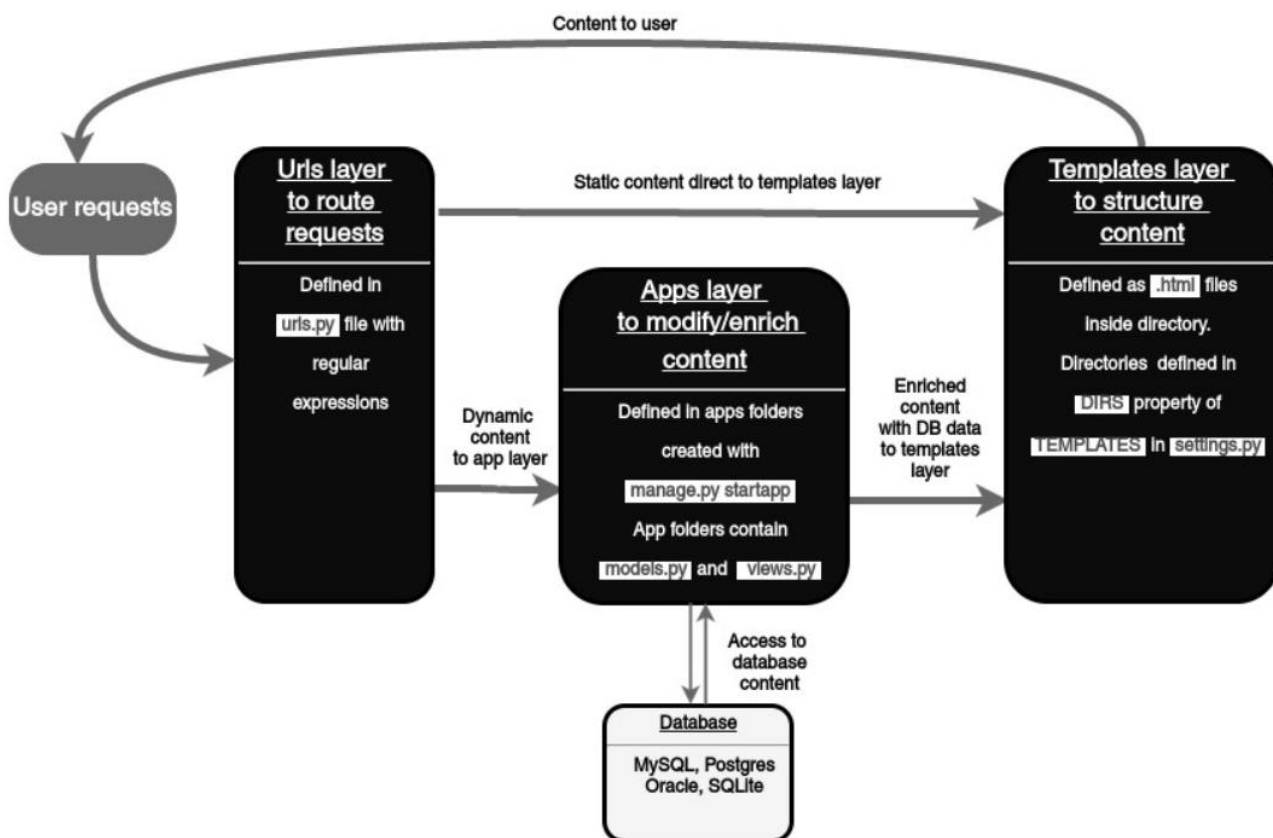


Рис. 3.4 – Взаємодія компонентів застосунку

Додатки (Django applications) створюються за допомогою команди *django-admin startapp <app_name>*. Вони початково містять мінімальний набір необхідних файлів для коректного функціонування. За бізнес-логіку обробки користувацьких запитів відповідають функції представлень (*views*), які прийнято розміщувати у спеціалізованому файлі *views.py* кожного окремого додатку. Для повноцінної роботи застосунку було розроблено такі основні обробники запитів:

- Авторизація / реєстрація користувача.
- Вихід з облікового запису.
- Зміна / скидання пароллю користувача.
- Підтвердження адреси електронної пошти та активація акаунту.
- Перегляд / зміна персональної інформації або даних про компанію.
- Доступ до адміністративної панелі додатку.
- Отримання набору проектів залежно від специфіки запиту.
- Створення / оновлення / видалення проекту в портфолію.
- Імпорт файлу із множиною описаних проектів.

- Керування унікальними сетами окремо обраних робіт.
- Поширення наборів власних проектів.
- Маніпуляція посиланнями спільного доступу.

Авторизація в застосунку можлива двома способами: стандартний – перевірка логіну й паролю попередньо створеного облікового запису; або ж за допомогою інтеграції зовнішньої системою LinkedIn – підтвердження секретного ключа та отримання ідентифікатора користувача, після чого вхід у систему використовуючи дані зі сторонньої платформи.

Кешування даних є найкращим підходом до швидкої відповіді на запити клієнтського браузера. Це забезпечує низьку кількість звернень до бази даних, запобігаючи перевантаженню БД. Стандартна структура кешування зображена на рисунку 3.5.

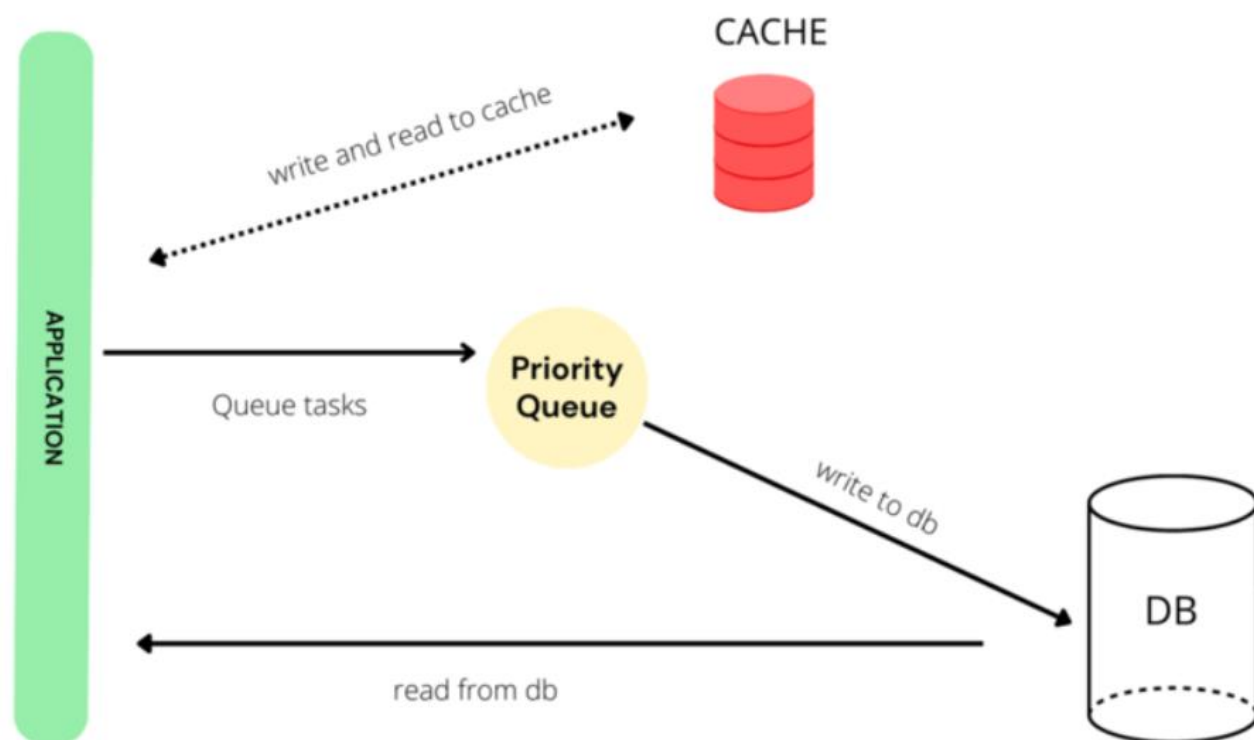


Рис. 3.5 – Альтернатива відтворення даних з кешу веб-застосунку

Django надає інструмент, який може використовувати різні механізми кешування, такі як Memcached і Redis, що дозволяє уникнути повторного виконання запитів.

Memcached – це проста, але потужна система з відкритим кодом у пам'яті, яка гарантовано повертає кешовані результати менш ніж за мілісекунди. *Memcached* також досить легко масштабована.

З іншого боку, *Redis* також є системою кешування з повністю доступним вихідним кодом, але пропонує більше функцій та можливостей використання. Саме тому у проекті було вирішено використати саме ***Redis***. Обробники автономних підпрограм використовують уже кешовані дані, що означає, що більшість запитів навіть не потрапляють до БД.

Сеанси користувача мають зберігатися в кеші застосунку Django, а оскільки *Redis* зберігає дані на дисках, усі сеанси для зареєстрованих користувачів надходитимуть не з бази даних, а з кешу. Щоб увімкнути кешування запитів, необхідно початково встановити ПЗ *Redis* та відповідну Python бібліотеку для коректної взаємодії. Після чого достатньо лише додати відповідну конфігурацію у файл налаштувань Django-проекту (рисунок 3.6).

```
CACHES = {
    "default": {
        "BACKEND": "django.core.cache.backends.redis.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379",
    },
    "select2": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/2",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
}
```

Рис. 3.6 – Конфігурація *Redis* для кешування запитів

3.3. Робота з базою даних

Вбудована ORM – дуже потужна особливість фреймворку, що надає елегантний спосіб взаємодії з базою даних. Це такий рівень абстракції, що дозволяє швидко оперувати всіма необхідними інструкціями для легкої

маніпуляції БД, використовуючи високорівневі структури коду Python. Зрештою, Django ORM перетворює усі команди в оператори мови SQL.

Підтримка постійного підключення значно пришвидшує роботу застосунку, оскільки не треба щоразу при будь-якому новому запиті повторно виконувати з'єднання з БД. Django ініціює підключення до бази даних одразу після першого запиту певної інформації з неї, після чого підтримує з'єднання для подальших запитів. Існує спеціальне налаштування часу підключення `CONN_MAX_AGE`, яке вказує фреймворку, коли необхідне закрити з'єднання з базою. Якщо виникає помилка в процесі обробки певного запиту, Django розриває існуюче з'єднання з БД та відкриває нове для наступних запитів. Таким чином, внутрішня помилка на стороні бази даних ніяк не вплине на виконання усіх інших запитів.

За замовчуванням, після встановлення фреймворк Django використовує базу даних *SQLite*. Такий вибір розробників зумовлений тим, що це дуже проста у використанні БД, яку майже не потрібно налаштовувати для початку розробки проекту, а також потребує мало ресурсів та може працювати на будь-якій мобільній платформі. Втім, для вагомих проектів вона зовсім не підходить, тому використаємо попередньо визначену ***PostgreSQL***.

Конфігурація підключення до баз даних визначається в головному файлі налаштувань Django-проекту. У нашому випадку це виглядає так:

```
DATABASES = {
    'default': dj_database_url.config(default=config('DATABASE_URL')),
}
```

Було прийняте рішення об'єднати та винести налаштування для з'єднання з БД у окрему змінну оточення в цілях безпеки, а також для простої організації визначення в одному місці усіх необхідних ключів для зв'язку із СУБД.

Метод `config()` бібліотеки `dj_database_url` дозволяє отримати словник підключення до бази даних, заповнений усіма даними, вказаними в URL-адресі.

Таким чином потрібно лише декларувати змінну оточення `DATABASE_URL`, що матиме такий вигляд:

postgres://USER:PASSWORD@HOST:PORT/DBNAME

Вона містить основні параметри для успішного з'єднання з БД:

- **postgres** – буде використаний двигун даної БД для підключення;
- **USER** – ім'я користувач, з даними якого виконується з'єднання;
- **PASSWORD** – пароль цього користувача;
- **HOST** – адреса сервера, на якому запущена база даних;
- **PORT** – порт для доступу до БД;
- **DBNAME** – назва створеної бази даних.

Кожен зі створених додатків проекту (Django apps), окрім *core*, описує декілька основних моделей, що в подальшому використовуватимуться для ініціалізації таблиць бази даних та дозволять оперувати даними з БД, використовуючи високорівневі структури.

Хоч за замовчуванням Django і надає вбудовану модель *User* для маніпулювання інформацією про користувача, але було вирішено створити власну, більш гнучку модель з розширеним набором необхідних полів, а також втілити процес аутентифікації, використовуючи електронну адресу:

```
class User(AbstractBaseUser, PermissionsMixin):
    """
    A model which implements the authentication model.

    Email and password are required. Other fields are optional.

    Email field are used for logging in.
    """
    email = models.EmailField(_('Email'), max_length=255, unique=True)
    name = models.CharField(_('Full name'), max_length=255)

    is_staff = models.BooleanField(
        _('staff status'),
        default=False,
        help_text=_('Designates whether the user can log into this admin
site.'),
    )
    is_active = models.BooleanField(
        _('active'),
        default=True,
        help_text=_(
            'Designates whether this user should be treated as active. '
            'Unselect this instead of deleting accounts.'
        ),
    ),
```

```

    )
    email_confirmed = models.BooleanField(
        _('email verified'),
        default=False,
        help_text=_('Designates whether the user has verified the email
address.'),
    )

    date_joined = models.DateTimeField(_('Date joined'), default=timezone.now)

    objects = UserManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['name']

    class Meta:
        verbose_name = _('User')
        verbose_name_plural = _('Users')
        ordering = ['name', '-date_joined']

    def get_first_name(self):
        chunks = self.name.split()
        if len(chunks) >= 1:
            return chunks[0]
        else:
            return ''

    @property
    def first_name(self):
        return self.get_first_name()

    def get_last_name(self):
        chunks = self.name.split()
        if len(chunks) >= 2:
            return chunks[1]
        else:
            return ''

    @property
    def last_name(self):
        return self.get_last_name()

    def __str__(self):
        return self.name

    def get_email_md5_hash(self):
        import hashlib
        m = hashlib.md5(self.email.lower().encode('utf-8')).hexdigest()
        return m

    def has_usable_password(self) -> bool:
        return super().has_usable_password()
    has_usable_password.boolean = True

    @property
    def days_on_site(self):
        from django.utils.timezone import now

```

```
delta = now() - self.date_joined
return delta.days
```

Також розроблений відповідний клас *UserManager*, що відповідає за створення та збереження користувача із вказаними обліковими даними й правами доступу:

```
class UserManager(BaseUserManager):
    """
    Creates and saves a User with the given email, name, password and optional
    extra info.
    """
    def _create_user(self, email,
                     name,
                     password,
                     is_staff, is_superuser, **extra_fields):
        """
        Creates and saves a User with the given username, email and password.
        """
        now = timezone.now()

        if not email:
            raise ValueError(_('The given email must be set'))

        email = self.normalize_email(email)
        user = self.model(
            email=email,
            name=name or '',
            is_staff=is_staff, is_active=True,
            is_superuser=is_superuser,
            date_joined=now,
            last_login=now,
            **extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_user(self, email, name, password=None, **extra_fields):
        return self._create_user(email, name, password, False, False,
        **extra_fields)

    def create_superuser(
        self, email, name, password=None, **extra_fields):
        """
        Creates and saves a superuser with the given email,
        name and password.
        """
        return self._create_user(email, name, password, True, True,
        **extra_fields)

    def get_by_natural_key(self, email):
        return self.get(email__iexact=email)
```

Кожен користувач представляє якусь конкретну корпорацію, а тому має відношення до таблиці компаній, що представлена моделлю *Company*:

```
class Company(models.Model):
    name = models.CharField(_('Company name'), max_length=150)
    logo = models.ImageField(_('Company logo'), upload_to='company_logos/')
    founder = models.OneToOneField(User, on_delete=models.CASCADE,
related_name='founder_company')
    year_founded = models.IntegerField(validators=[MinValueValidator(1950),
MaxValueValidator(datetime.now().year)])
    website = models.URLField(_('Company website'))
    email = models.EmailField(_('Contact email'), max_length=255, unique=True)
    slogan = models.CharField(_('Company slogan'), max_length=255)
    description = models.TextField(_('Description'))

class Meta:
    verbose_name_plural = _('Companies')

def __str__(self):
    return f"{self.name}"

def save(self, *args, **kwargs):
    """Delete the previous logo from the storage if a new one has been
uploaded."""

    try:
        company = Company.objects.get(pk=self.pk)
        if company.logo != self.logo:
            company.logo.delete(save=False) # `save=False` to prevent a
recursive save
    except:
        pass
    super().save(*args, **kwargs)
```

Ключовою моделлю для збереження даних портфолію виступає модель *Project*. Вона має основні поля для достатньо точного опису виконаного проекту, може містити по декілька індустрій застосування та технологій розробки, а ще посилається на відповідного користувача для контролю доступу до об'єкта. Тут також реалізована логіка взаємодії зі стороннім інструментом, що поліпшує продуктивність обробки деяких клієнтських запитів. Під час будь-якої зміни та збереження інформації в *PostgreSQL*, відбувається синхронізації даних із *ElasticSearch*. Модель була визначена наступним чином:

```
class Project(models.Model):
    title = models.CharField(_('Project name'), max_length=150)
    description = models.TextField(_('Description'))
    author = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='author_projects')
    industries = models.ManyToManyField(Industry,
```

```

related_name='projects_containing_industry')
technologies = models.ManyToManyField(Technology,
related_name='projects_containing_technology')
url = models.URLField(_('Project website'), null=True, blank=True)
url_is_active = models.BooleanField(_('Website is active'), default=False)
created_at = models.DateTimeField(_('Date created'), auto_now_add=True)
updated_at = models.DateTimeField(_('Date updated'), auto_now=True)
is_private = models.BooleanField(_('Project is private'), default=True)
is_original = models.BooleanField(_('Project is original'), default=True)

def get_elasticsearch_document(self):
    doc = {
        'title': self.title,
        'description': self.description,
        'author': self.author.id,
        'project_id': self.id,
        'is_private': self.is_private,
        'industries': list(self.industries.values_list("id", flat=True)),
        'technologies': list(self.technologies.values_list("id",
flat=True)),
    }
    return json.dumps(doc)

def save(self, *args, **kwargs):
    refresh_index = kwargs.pop('refresh_index', True)
    # Control admin import csv. Do not create elastic documents if
    # transaction will rollback (first dry_run import to preview changes)
    dry_index_update = kwargs.pop('dry_index_update', False)

    self._refresh_index = refresh_index
    self._dry_index_update = dry_index_update # to control signal
m2m_changed

    super().save(*args, **kwargs)

    if not dry_index_update:
        from .utils import update_elastic_document
        update_elastic_document(self, refresh_index=refresh_index)

def __str__(self):
    return f"{self.title}"

class Meta:
    verbose_name_plural = _('Projects')

```

Також слід зазначити ще дві моделі даних, що є основними у підході до створення застосунку. Мова йде про *Set* та *SetSharedLink*. Відповідні класи були описані для доповнення функціоналу застосунку. Завдяки цьому з'являється можливість створення спеціалізованих наборів робіт із власного портфолію та управління посиланнями спільного доступу до таких сетів. Ці моделі описані нижче.

```

class Set(models.Model):
    name = models.CharField(_('Set name'), max_length=150)
    author = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='author_sets')
    projects = models.ManyToManyField(Project,
related_name='sets_containing_project')
    created_at = models.DateTimeField(_('Date created'), auto_now_add=True)
    updated_at = models.DateTimeField(_('Date updated'), auto_now=True)

    def __str__(self):
        return f'{self.name}'

class Meta:
    verbose_name_plural = 'Sets'
    unique_together = ('name', 'author')
    ordering = ['-id']

class SetSharedLink(models.Model):
    set = models.ForeignKey(Set, on_delete=models.CASCADE,
related_name='set_shared_link')
    token = models.CharField(_('Shared link'), max_length=255, unique=True)
    ip_addresses = ArrayField(models.CharField(max_length=255, blank=True),
default=list)
    link_open_counter = models.PositiveIntegerField(default=0)
    created_at = models.DateTimeField(_('Date created'), auto_now_add=True)

    def __str__(self):
        return f'Shared link for Set "{self.set.name}"'

class Meta:
    verbose_name_plural = _('Sets Shared Links')

```

Додатково у класі *SetSharedLink* були визначені атрибути *ip_addresses* та *link_open_counter*. Це зроблено з метою моніторингу активності посилань спільного доступу та подальшого аналізу аналітики поширених портфолію. Також слід зазначити, що IP-адреси користувачів будуть попередньо оброблені та збережені лише у вигляді *хешу* з метою покращення безпеки даних.

3.4. Інтеграція Elasticsearch

ElasticSearch — це система розподіленого пошуку та аналітики, створена на основі *Apache Lucene*. З моменту свого випуску в 2010 році *ElasticSearch* швидко став найпопулярнішою пошуковою системою і зазвичай використовується для аналізу журналів логування, повнотекстового пошуку, аналітики безпеки, бізнес-аналітики та операційної аналітики.

ElasticSearch розроблений для пошуку та аналітики всіх типів даних, включаючи текстові, числові, геопросторові, структуровані та неструктуровані. Відомий своїми простими протоколами взаємодії *REST API*, розподіленістю, швидкістю та масштабованістю, *ElasticSearch* є центральним компонентом *ElasticStack* – набору безкоштовних і відкритих інструментів для прийому, збагачення, зберігання, аналізу та візуалізації даних.

ElasticSearch – це сучасний пошуковий двигун, що побудований на Java, має повністю відкритий вихідний код та позиціонується як *NoSQL* база даних. Однак, на відміну від більшості *NoSQL* систем, *ElasticSearch* зосереджений на широких можливостях і функціях пошуку настільки, що найпростіший спосіб отримати дані з ES – це шукати їх за допомогою обширного *ElasticSearch API*. У контексті аналізу даних *ElasticSearch* використовується разом з іншими компонентами в *ELK Stack*: *Logstash* і *Kibana*, і виконує роль індексатора та зберігача даних.

Налаштування *ElasticSearch* виконуються за допомогою спеціального файлу конфігурації, розташування якого залежить від операційної системи. У цьому файлі вказуються загальні параметри (наприклад, ім'я вузла); налаштування мережових підключень (наприклад, хост і порт); зазначається, де саме зберігати дані індексу; надається дозвіл на використання певного об'єму оперативної пам'яті; описується місцезнаходження файлів журналів тощо.

Необроблені дані надходять в *ElasticSearch* з різних джерел, включаючи log-журнали, системні показники та веб-додатки. Приймання даних – це процес, за допомогою якого ці необроблені дані аналізуються, нормалізуються та збагачуються перед індексацією в *ElasticSearch*. Після індексації інформації в системі двигуна можна виконувати складні запити до даних і використовувати агрегації для отримання складних результатів

Індекс – це набір документів, пов'язаних один з одним. *ElasticSearch* зберігає дані як документи JSON. Кожен документ співвідносить набір ключів (імен, полів або властивостей) із відповідними значеннями (рядами, числами,

логічними значеннями, датами, масивами, геолокаційними точками або іншими типами даних).

ElasticSearch використовує структуру даних, яка називається інвертованим індексом, що розроблена для забезпечення дуже швидкого повнотекстового пошуку. Обернений індекс перелічує кожне унікальне слово, яке з'являється в будь-якому документі й ідентифікує інші документи, у яких зустрічається схоже слово.

Під час процесу індексування ElasticSearch зберігає документи та створює інвертований індекс, щоб зробити пошук даних документа майже в режимі реального часу. Індексування ініціюється за допомогою API, завдяки якому можливо додавати або оновлювати JSON- документи у певному індексі [31].

Структура типового документу для збереження проектів портфоліо у створеному ElasticSearch індексі має такий вигляд:

```
{
  "mappings": {
    "properties": {
      "title": {"type": "text"},
      "description": {"type": "text"},
      "author": {"type": "keyword"},
      "project_id": {"type": "long"},
      "is_private": {"type": "boolean"},
      "industries": {"type": "keyword"},
      "technologies": {"type": "keyword"}
    }
  }
}
```

У процесі розробки веб-застосунку з метою поліпшення продуктивності програми був створений такий базовий запит, що пізніше буде доповнюватись агрегаційними функціями для втілення процесу повнотекстового пошуку та фасетної навігації.

```
# ElasticSearch query
author_filter = {"terms": {"author": [request.user.id]}}
public_or_author_filter = [public_projects_filter or author_filter]
query = {
  "from": (int(request.GET.get('page', 1)) - 1) * settings.PAGE_SIZE,
  "size": settings.PAGE_SIZE,
  "sort": [
    {"project_id": {"order": "desc"}}
  ],
  "query": {
    "bool": {
```

```

        "must": public_or_author_filter
    },
    "aggs": {
        "industries": {
            "terms": {
                "field": "industries",
                "size": 1000
            }
        },
        "technologies": {
            "terms": {
                "field": "technologies",
                "size": 1000
            }
        },
        "unfiltered_industries": {
            "global": {},
            "aggs": {
                "all_industries": {
                    "aggs": {
                        "industries": {
                            "terms": {"field": "industries", "size": 1000}
                        }
                    },
                    "filter": {
                        "bool": {
                            "must": public_or_author_filter
                        }
                    }
                }
            }
        },
        "unfiltered_technologies": {
            "global": {},
            "aggs": {
                "all_technologies": {
                    "aggs": {
                        "technologies": {
                            "terms": {"field": "technologies", "size": 1000}
                        }
                    },
                    "filter": {
                        "bool": {
                            "must": public_or_author_filter
                        }
                    }
                }
            }
        }
    }
}

```

Переваги використання *ElasticSearch*:

- **Швидкий та зручний процес взаємодії.** Elasticsearch пропонує простий API на основі REST архітектури, простий HTTP-інтерфейс та використовує документи JSON без схем, що полегшує початок роботи та швидке створення програм для різноманітних випадків використання.
- **Висока продуктивність.** Розподілений характер обчислювальної системи Elasticsearch дає змогу обробляти великі обсяги даних паралельно, швидко знаходячи найкращі відповідності конкретним запитам.
- **Безкоштовні інструменти та плагіни.** Elasticsearch інтегровано з *Kibana* – популярним інструментом візуалізації та звітності. Він також пропонує інтеграцію з *Beats* і *Logstash*, а ще дозволяє легко трансформувати вихідні дані та завантажувати їх у кластер. Існує можливість використовувати ряд плагінів Elasticsearch з відкритим вихідним кодом, таких як аналізатори мови та підказки, щоб додати більшу функціональність до розроблених застосунків.
- **Операції майже в реальному часі.** Операції Elasticsearch, такі як читання або запис даних, зазвичай займають менше секунди. Це дає змогу використовувати Elasticsearch майже в режимі реального часу, наприклад, для моніторингу додатків і виявлення аномалій.
- **Легка розробка програм.** Elasticsearch може використовуватись майже будь-де, оскільки підтримує різні мови програмування, як-от: *Java*, *Python*, *PHP*, *JavaScript*, *Ruby* та багато інших.

3.5. Сервіс для відкладених задач

Розробка функціоналу для виконання фонових задач проводилася за допомогою бібліотеки *Celery*. Це гнучка, асинхронна, розподілена система для обробки встановленої черги завдань, спілкування з якою відбувається за допомогою повідомлень. Принцип роботи показаний на рисунку 3.7.

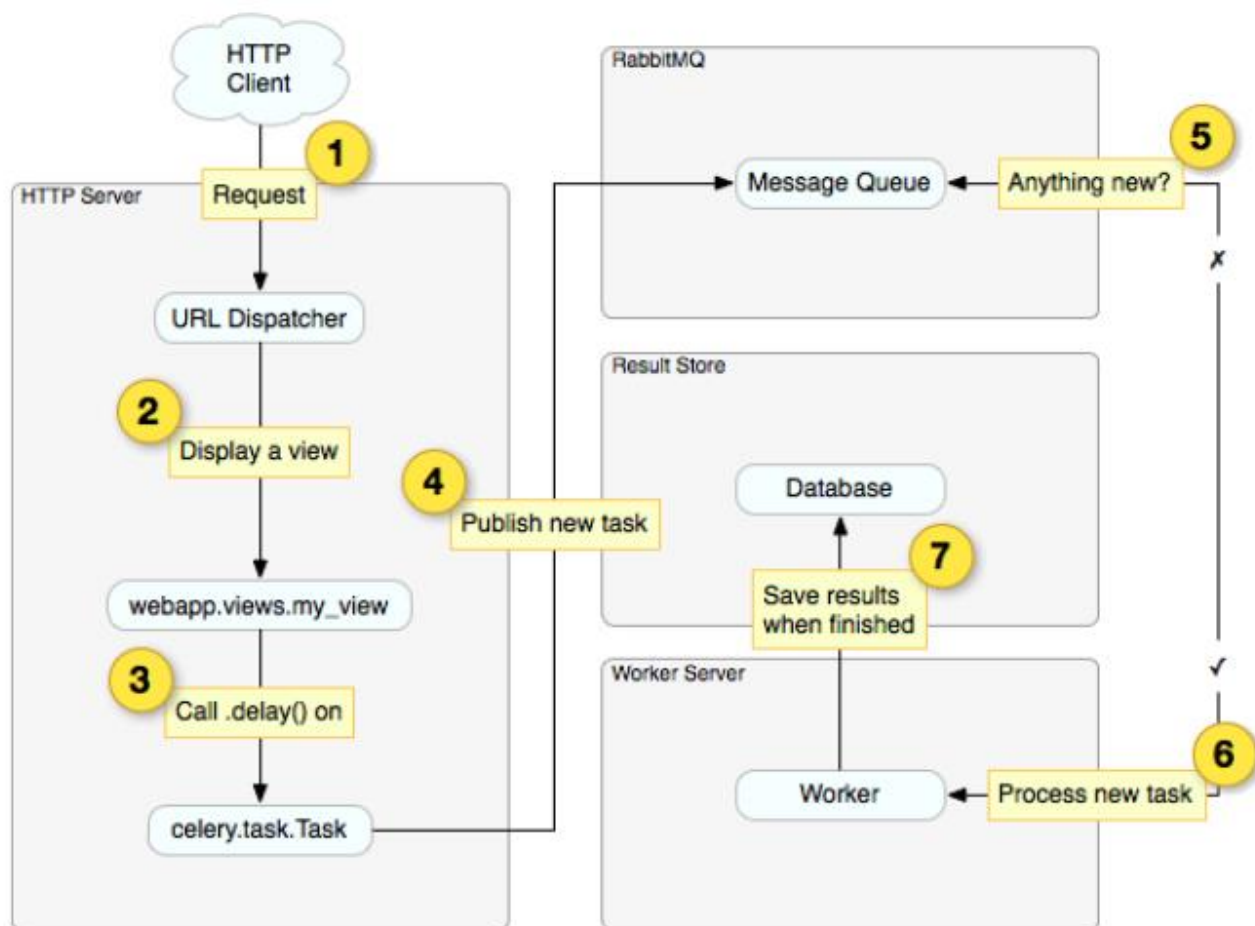


Рис. 3.7 – Принцип взаємодії веб-сервера з Celery

Головними перевагами Celery є:

- асинхронне виконання відкладених задач;
- виконання періодичних задач на вимогу;
- розподілене по окремих серверах або ж конкурентне виконання задач в межах виділеного процесу;
- моніторинг стану виконання задач;
- автоматичне повторне виконання задач у разі виникнення помилки;
- повне логування усіх подій;
- чіткі повідомлення про помилки.

Конфігурація *Celery* відбувається дуже просто, достатньо лише ініціалізувати екземпляр *Celery* та коротко описати властивості в спеціальному модулі *celery.py* в директорії головного додатку проекту. Визначення усіх

відкладених завдань виконується в окремому файлі *tasks.py*, що знаходиться в кожному Django application. Приклад однієї з таких задач показаний нижче.

```
@app.task(bind=True, default_retry_delay=1 * 60)
def send_email_to_user(self, mail_subject: str, plain_message: str, send_to:
list[str],
                        html_message: str = None) -> None:
    """Celery task to send email to user."""

    try:
        send_mail(
            subject=mail_subject,
            message=plain_message,
            from_email=settings.DEFAULT_FROM_EMAIL,
            recipient_list=send_to,
            html_message=html_message,
            fail_silently=False,
        )
    except smtplib.SMTPException as ex:
        self.retry(exc=ex)
```

Функціями для виконання у вигляді відкладених задач було реалізовано такі ключові довготривалі завдання:

1. Формування й збереження даних у БД та індекс Elasticsearch про проекти портфолію, що надходять за допомогою парсингу спеціального завантаженого файлу.
2. Надсилання листів на поштові адреси користувачів для активації акаунтів, зміни та підтвердження персональних ключів доступу, отримання статистики щодо посилок спільного доступу.
3. Створення резервних копій БД.

Для моніторингу та адміністрування відкладених задач використовувався додатковий інструмент **Flower**, що надає власний зручний інтерфейс взаємодії (рисунок 3.8) [32].

Celery Flower						
Workers						
☐ Shut Down						
	Name	Status	Concurrency	Completed Tasks	Running Tasks	Queues
☐	celery1.pi.local	Online	4	13902	0	images, data, video
☐	celery2.pi.local	Online	4	13900	0	images, data, video
☐	celery3.pi.local	Online	4	13826	0	images, data, video
☐	celery4.pi.local	Online	1	1989	0	data
☐	celery5.pi.local	Online	1	1983	0	data
☐	celery6.pi.local	Offline	3	2245	3	
☐	celery7.pi.local	Online	3	2283	3	celery, data
☐	celery8.pi.local	Online	3	2279	3	celery
☐	celery9.pi.local	Online	3	2287	3	celery

Рис. 3.8 – Інтерфейс додатку Flower

Основними перевагами *Flower* є:

- моніторинг процесу виконання задач у режимі реального часу;
- можливість детального огляду задачі (аргументи, момент старту, час обробки задачі та ін.);
- огляд статусу *воркерів* та задач;
- зупинення чи перезапуск задач або ж процесів, що їх виконують;
- масштабування та контроль ресурсів *воркерів*;

3.6. Реалізація клієнтської частини

Розробка інтерфейсу користувача виконувалась на основі *Django template language* за допомогою фреймворків *Bootstrap* та *JQuery*. Було спроектовано та реалізовано усі необхідні компоненти веб-сторінок, що попередньо планувалось використати у візуальному відображенні застосунку. Інтерфейс веб-додатку досить простий, інтуїтивно зрозумілий та повністю покриває задуманий функціонал програми. Далі буде розглянуто лише ключові сторінки.

На рисунку 3.9 бачимо домашню сторінку застосунку, яка повідомляє базову інформацію та надає можливість вибору користування системою в залежності від ролі користувача.

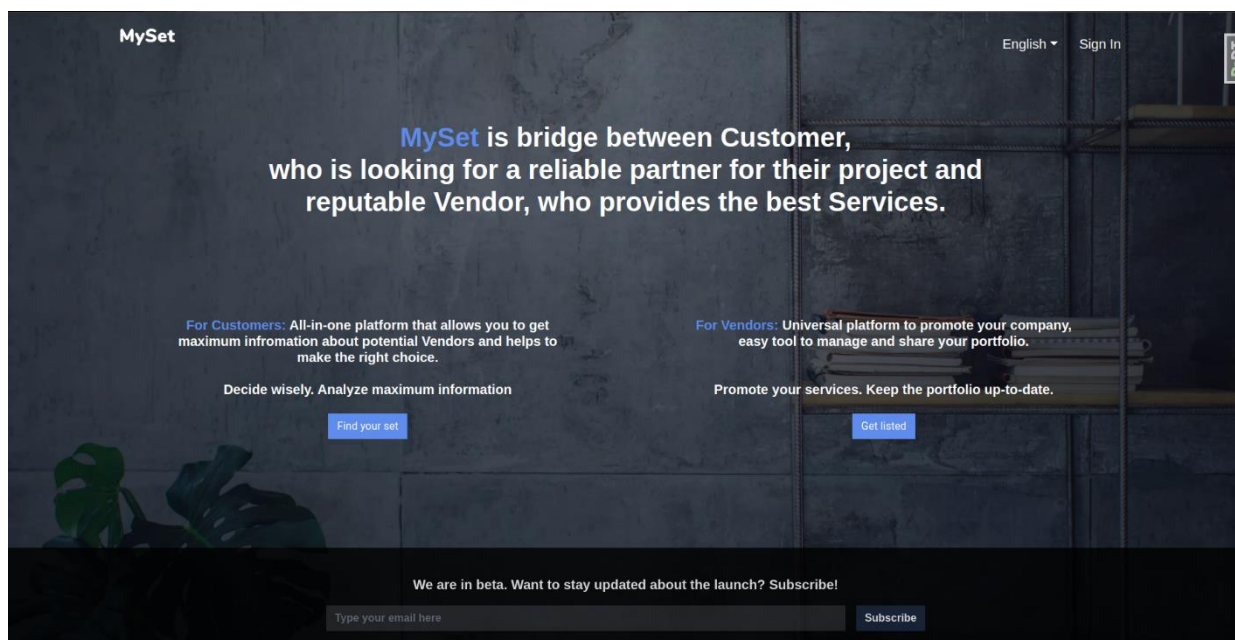


Рис. 3.9 – Домашня сторінка застосунку

На рисунку 3.10 представлена сторінка аутентифікації користувача. Ще тут є відповідна кнопка реєстрації нового облікового запису, а також можливий вхід у систему за допомогою *LinkedIn*.

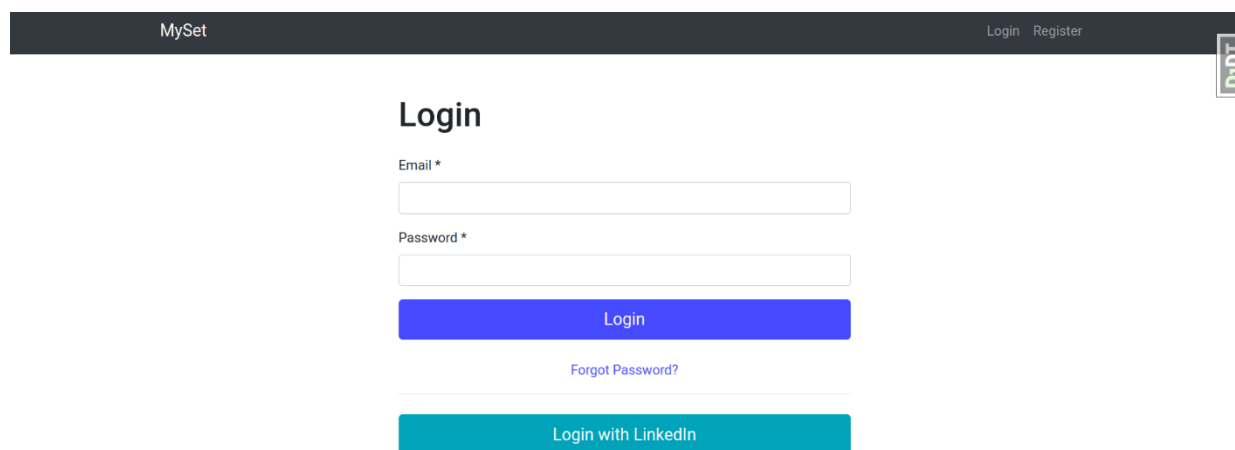


Рис. 3.10 – Форма аутентифікації

Після входу в систему потрапляємо на головну сторінку маніпулювання власним портфолію (попередньо для тестування були згенеровані записи в БД з **випадковими** назвами, описом, індустріями, технологіями, посиланнями і т.п.).

Тут наявний необхідний функціонал для додавання, зміни чи видалення вже існуючих проектів, їх фільтрування або ж текстового пошуку конкретного серед великої кількості. Також наявні інші вкладки для управління публічним портфоліо та спеціалізованими наборами окремих робіт (рисунок 3.11).

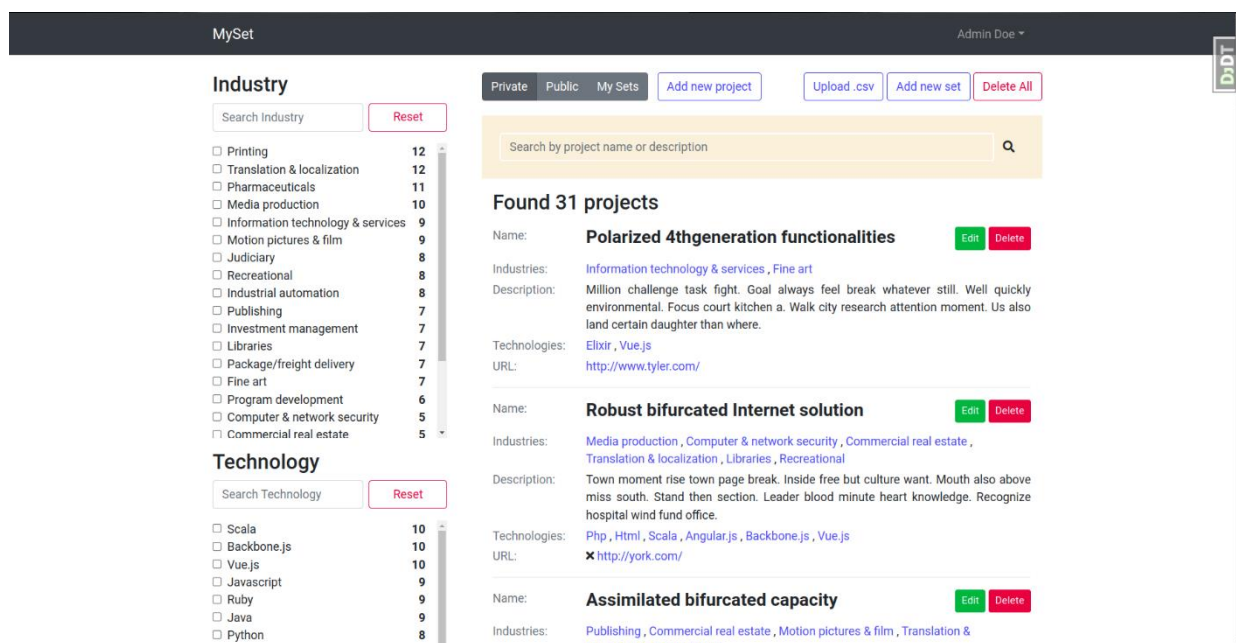


Рис. 3.11 – Головна сторінка управління портфоліо

Існує можливість додавання нових робіт до портфоліо як по одній, заповнивши поля вручну (рисунок 3.12), так і автоматичне створення/оновлення одразу декількох, завантаживши відповідний файл (рисунок 3.13).

Рис. 3.12 – Створення нового проекту в портфоліо

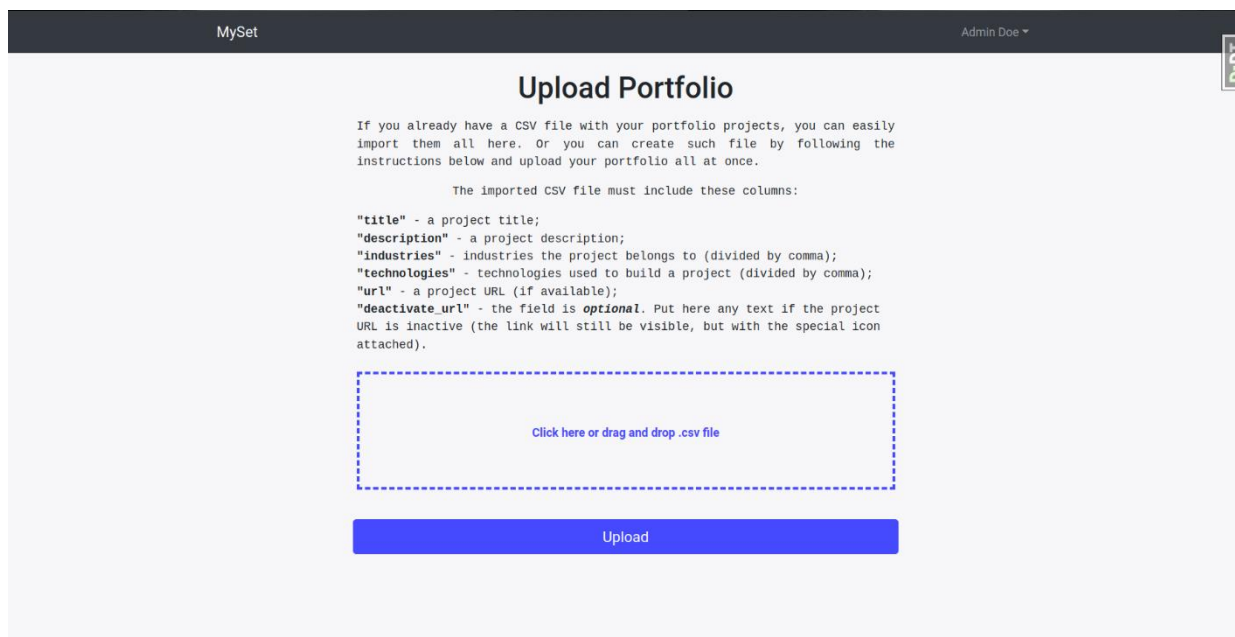


Рис. 3.13 – Автоматичне додавання одразу декількох проектів

На рисунку 3.14 можна спостерігати процес створення та наповнення нового *сету* обраними проектами з портфолію.

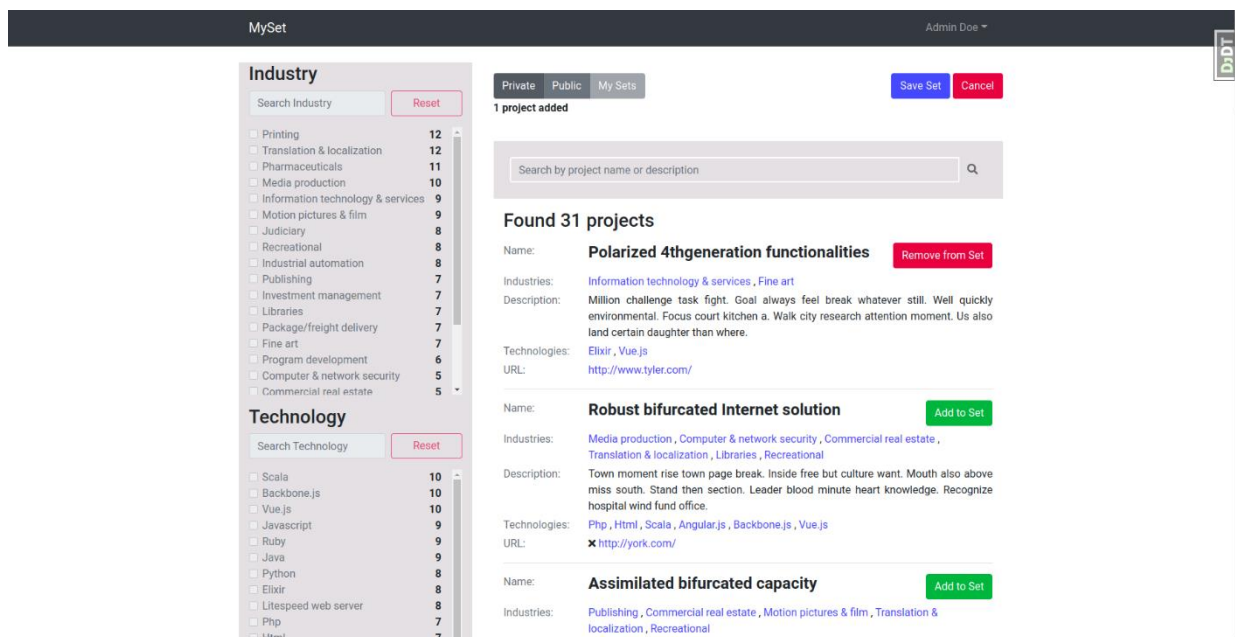


Рис. 3.14 – Створення окремого набору робіт

Сторінка маніпуляції спеціалізованими *сетами* робіт зображена на рисунку 3.15. Тут можна створити, змінити чи видалити окремий набір, дублювати та переназвати його (наприклад, для іншого клієнта), згенерувати та поділитись посиланням спільного доступу на окремий *сет* або ж відправити інформацію електронною поштою.

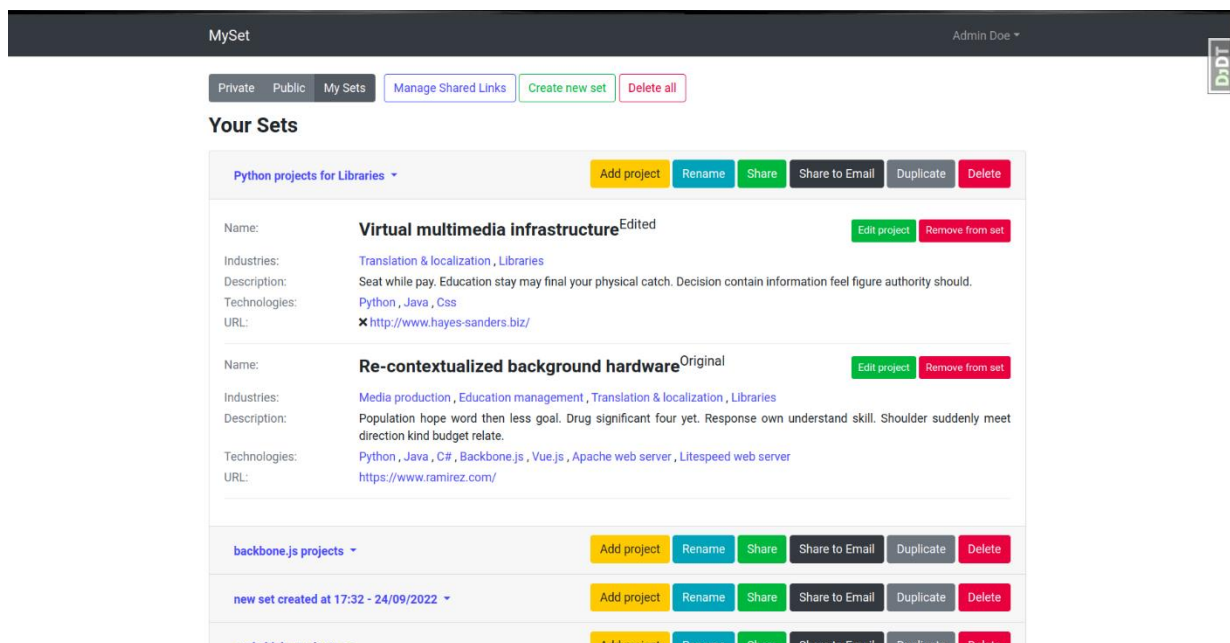


Рис. 3.15 – Сторінка управління наборами проектів

Попередньо при першому вході користувач заповнює відповідну форму, вводячи інформацію про власну ІТ-компанію для подальшого її відображення на усіх створених наборах проектів цього користувача. Посилання спільного доступу вказує на детальну сторінку з публічним портфоліо, створеним для того чи іншого клієнта (рисунк 3.16).

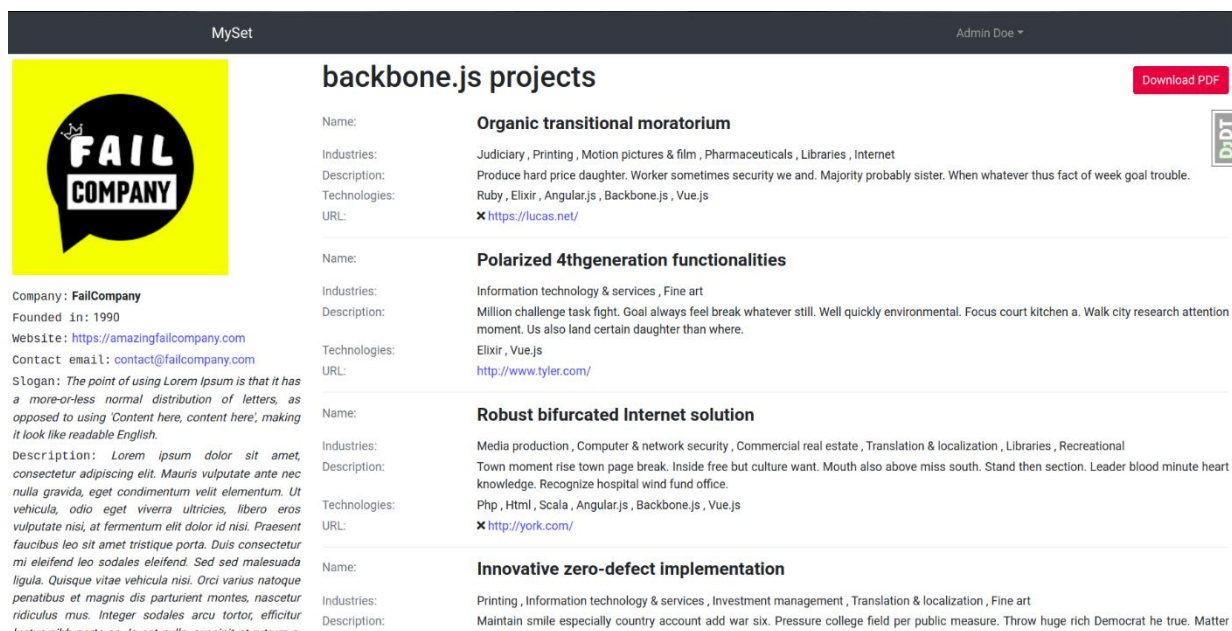


Рис. 3.16 – Публічна сторінка окремого набору з портфоліо

На додаток до основного функціоналу з кастомним інтерфейсом була налаштована адміністративна панель Django, що також має усі ті самі функції,

але надає більше контролю над різними об'єктами. Інтерфейс такого інструменту зображено на рисунку 3.17.

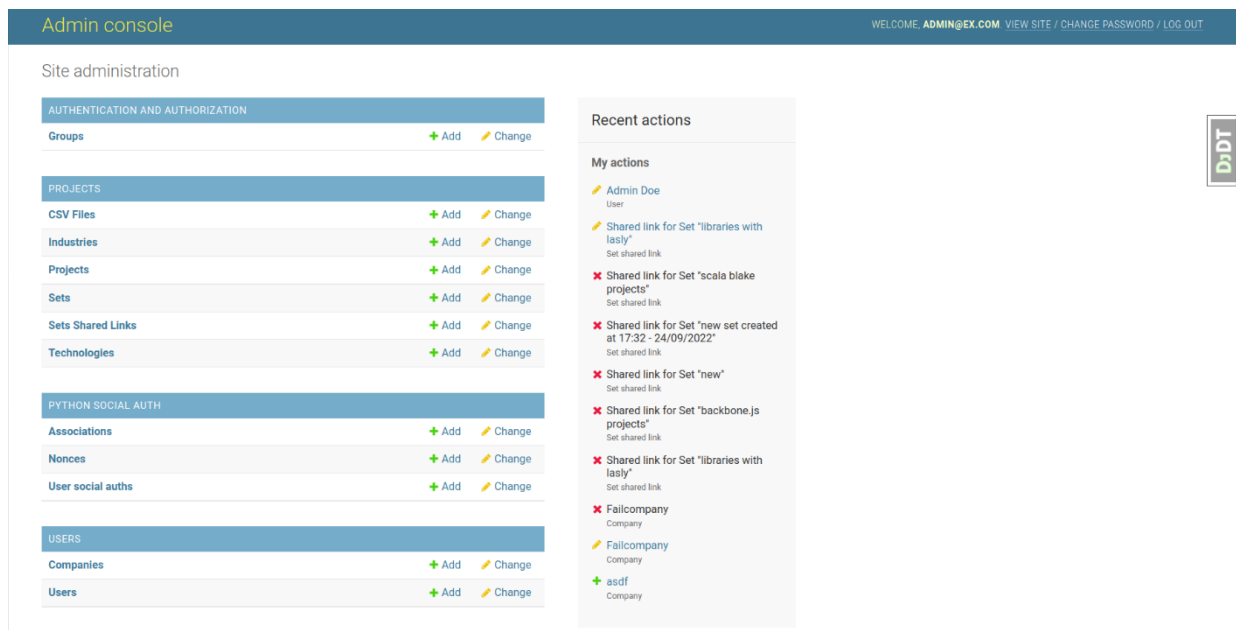


Рис. 3.17 – Інтерфейс адміністративної панелі застосунку

ВИСНОВКИ ДО РОЗДІЛУ 3

У цьому розділі був описаний процес проектування й реалізації бізнес-логіки та користувацького інтерфейсу створеного застосунку. Було розглянуто принцип роботи сервера, його функціональні особливості та взаємодію основних компонентів системи.

Розроблено набір функцій-обробників, що відповідають на запити користувача, впроваджено систему авторизації для безпечності та контролю доступу до ресурсу. Описано структуру БД та основні методи взаємодії для надійного збереження даних.

Для збільшення продуктивності системи було вирішено розгрузити основний веб-сервер та виконувати довготривалі операції у фоновому режимі за допомогою асинхронної бібліотеки *Celery*.

Функціонал фільтрування та повнотекстового пошуку було покладено на спеціальний новітній двигун. Проект сконфігуровано так, що синхронізація даних з *ElasticSearch* відбувається при будь-яких змінах в основній БД.

Розробка інтерфейсу користувача проводилась згідно з обраних технологій у попередньому розділі. Були представлені відповідні скріншоти з прикладами роботи застосунку.

Кожен етап проектування й реалізації системи проводився згідно з поставленими вимогами, що були зазначені в технічному завданні. У результаті розробки був створений веб-застосунок, що надає унікальний графічний інтерфейс та своїм функціоналом повністю задовольняє поставлені задачі.

РОЗДІЛ 4

РОЗРОБКА СТАРТАП ПРОЕКТУ

4.1. Опис ідеї проекту

Таблиця 4.1

Опис ідеї проекту		
Зміст ідеї	Напрями застосування	Вигоди для користувача
Система управління портфоліо ІТ-компанії для залучення нових клієнтів	Керування приватним та публічним портфоліо компанії	Можливість зручного контролю портфоліо власної ІТ-компанії
	Гнучке налаштування	Автоматизація процесу створення/оновлення проектів у портфоліо
	Допомога в питаннях маркетингу та сейлз відділу ІТ-компанії	Створення спеціалізованих наборів портфоліо для поширення серед клієнтів різних індустрій розробки ПЗ

Для реалізації ідеї проекту використовувались такі основні технології:

- мова програмування Python;
- фреймворк Django;
- СУБД PostgreSQL;
- пошуковий двигун Elasticsearch;
- розподілена асинхронна система виконання задач Celery;
- фронтенд фреймворки: Bootstrap, JQuery;
- середовище PyCharm.

4.2. Технологічний аудит проекту

Таблиця 4.2

Технологічна здійсненність ідеї проекту

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Система управління портфоліо	Мова програмування Python, фреймворк Django	Наявні	Доступні безкоштовно
2	ІТ-компанії для залучення нових клієнтів	СУБД PostgreSQL	Наявна	Доступна безкоштовно
3		Пошуковий двигун Elasticsearch	Наявна	Доступна безкоштовно з базовим функціоналом
4		Система відкладених задач Celery, система моніторингу виконання відкладених задач Flower	Наявні	Доступні безкоштовно
5		Мова програмування JavaScript, фреймворки Bootstrap та JQuery	Наявні	Доступні безкоштовно
6		Система моніторингу помилок Sentry	Наявна	Доступна безкоштовно з базовим функціоналом

Згідно з проведеними дослідженнями, було обрано технології, що постачаються безкоштовно у базовому функціоналі. Усі технології є повністю доступними.

4.3. Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів. В таблиці 4.3 наведена попередня характеристика потенційного ринку для розроблюваного стартап-проекту.

Таблиця 4.3

Попередня характеристика потенційного ринку стартап-проекту

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	2 (розробник, користувач Інтернет-ресурсу)
2	Динаміка ринку (якісна оцінка)	Проблема маркетингу власної ІТ-компанії зараз дуже розповсюджена, потреба в нових зручних системах управління та поширення портфоліо виконаних робіт постійно зростає
3	Наявність обмежень для входу (вказати характер обмежень)	Для комерційних цілей потрібно купувати підписку для використання певних технологій та наявна конкуренція
4	Специфічні вимоги до стандартизації та сертифікації	Специфічних вимог до стандартизації та сертифікації немає

Таблиця 4.4

Характеристика потенційних клієнтів стартап-проекту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Створення та редагування бази власних проектів чи досліджень в портфоліо	Розробники-початківці, корпоративні клієнти, дослідники	Сфера надання послуг фахівця та технології використання	Чіткий та зручний облік власних публічних чи приватних робіт в портфоліо
2	Контроль, налаштування та поширення окремих наборів робіт серед специфічних клієнтських аудиторій з метою збільшити впізнаваність бренду	Працівники маркетингового відділу або ж відділу продажів певної ІТ-корпорації	Особливості рекламної кампанії власної фірми	Формування окремих спеціально визначених наборів проектів для конкретного клієнта в тій чи іншій області розробки ПЗ

Таблиця 4.5

Фактори загроз стартап-проекту

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренція	На даний момент існують аналоги, які в основному повторюють функціонал одне одного, тому є загроза не зацікавити клієнта	Швидка реакція команди розробників на побажання потенційних клієнтів, постійне оновлення та розширення функціоналу
2	Недостатньо коштів для підтримки продукту	Задля успішного функціонування продукт потрібно підтримувати та розвивати, для чого необхідні як матеріальні, так і людські ресурси	Пошук можливих спонсорів
3	Несприятлива ситуація у країні – криза	Девальвація валюти, автоматичне підвищення ціни у національній валюті	Необхідно розробити додаткові стимули для покупців.
4	Несприятлива ситуація у країні – блекаут	Відключення електропостачання по всій території країни запуски проекту	Частіше створення резервних копій бази даних та стану застосунку

Таблиця 4.6

Фактори можливостей стартап-проекту

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Залучення інвестицій	Скласти чіткий бізнес-план, залучити інвесторів. Це дозволить розширити штат та закупити потрібну техніку	Прискорення виходу на ринок та покращення системи
2	Розширення функціоналу продукту	Зростання до рівня b2b системи	Доповнення можливостей потенційних користувачів для створення нової b2b платформи
3	Збільшення клієнтів-підприємців	Укладання договорів на співробітництво з підприємствами	Розширення базового функціоналу та створення преміальних можливостей використання
4	Збільшення клієнтів-користувачів	Залучення нових користувачів, ріст популярності	Реклама системи на різних ресурсах, проведення презентацій, акцій

Таблиця 4.7

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкуренто- спроможною)
Складність запуску застосунку	На ринку уже є досить успішні конкуренти	Якісна рекламна кампанія та удосконалення функціоналу
Популярність вже існуючих аналогів	Перехопити аудиторію користувачів у існуючих реалізацій	Експериментувати з ідеями для нових оновлень, впроваджувати нові рішення в мінімальний термін

Таблиця 4.8

Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти в галузі	Постачальники	Клієнти	Товари-замінники
	Coroflot, Flickr, Entsane, Crevado	Clutch.co, GoodFirms	Coroflot, Flickr, Entsane, Crevado	Професійні фотографи, дизайнери, дослідники, розробники	Наявність конкурентів, але прямих аналогів замінників не існує

Продовження таблиці 4.8

Висновки	Наявна велика кількість різноманітних рішень з боку конкурентів	При виході на рівень b2b системи, існують конкурентні гіганти в цій сфері	Постачальники підтримують вже створені проекти та додають новий функціонал, чим залучають все більше користувачів	Клієнтом може стати будь-який користувач мережі Інтернет, що зацікавлений в зручному керуванні портфелю власних проектів та розповсюдженні серед потенційних клієнтів	Обмежень через товари-замінники немає, але є сильна конкуренція з існуючими реалізаціями
----------	---	---	---	---	--

Таблиця 4.9

Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Більш лояльна цінова політика	Створення окремої цінової політики для менших клієнтів, для яких ціни продуктів конкурентів зависокі

Продовження таблиці 4.9

2	Вітчизняний продукт	Розвиток вітчизняного продукту, створення конкуренції європейським чи ще якимось аналогам
3	Автоматичне впровадження чи редагування власного портфоліо	Можливість автоматично доповнити портфоліо проектів завдяки попередньо створеному файлу, як це роблять багато компаній у наш час – ведуть записи про виконані раніше замовлення саме у формі таблиць
4	Розповсюдження унікальних <i>сетів</i> серед клієнтських мас	Керування множиною як приватних, так і публічних портфоліо завдяки функції створення окремих спеціалізованих наборів робіт. Рекламування бренду та залучення нових інвестицій за допомогою поширення спеціальних портфоліо клієнтам наближених сфер
5	Швидке реагування на відгуки	На етапі розробки неможливо передбачити всі нюанси застосування. Завдяки забезпеченню швидкого відгуку, можна використовувати користувачів у якості бета-тестувальників, що дозволить максимально швидко адаптувати додаток під потреби реальних клієнтів

Таблиця 4.10

Порівняльний аналіз сильних та слабких сторін

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у по- рівнянні з розроблюваним продуктом						
			-3	-2	-1	0	1	2	3
1	Більш лояльна цінова політика	11			+				
2	Вітчизняний продукт	12	+						
3	Автоматичне впровадження чи редагування власного портфоліо	16		+					
4	Розповсюдження унікальних <i>сетів</i> серед клієнтських мас	18	+						
5	Швидке реагування на відгуки	15					+		

Таблиця 4.11

SWOT-аналіз стартап-проекту

Сильні сторони:	Слабкі сторони:
– вітчизняний продукт; – автоматичне доповнення чи редагування портфоліо; – поширення окремих наборів; – швидке реагування на відгуки.	– велика кількість конкурентів; – потреба в інвесторах для підтримки стартап-проекту; – спрямованість додатку тільки на фахівців точних наук.

Продовження таблиці 4.11

Можливості:	Загрози:
– доповнення функціоналу; – вихід на ринок b2bресурсів; – впровадження системи на інші платформи.	– конкуренція та складність виходу на міжнародний ринок; – невдача в пошуку інвесторів; – критична ситуація в країні запуску проекту.

Таблиця 4.12

Альтернативи ринкового впровадження стартап-проекту

№	Альтернатива ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Активна реклама на популярних Інтернет-ресурсах та в соц. мережах	Висока. Ефективна рекламна кампанія допоможе завоювати популярність	Одразу після старту
2	Укладення договорів з ІТ-корпораціями	Середня. Заздалегідь не відомо, чи захочуть співпрацювати	Через півроку
3	Вихід на міжнародний рівень	Середня. Оскільки потрібно відкривати відділ для контролювання	Через півроку-рік

4.4. Розроблення ринкової стратегії проекту

Таблиця 4.13

Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Кваліфіковані ІТ-фахівці	Готові сприйняти	Продукт необхідний вже зараз	Висока конкуренція	Вхід не є складний
2	Приватні ІТ-компанії	Не всі компанії готові сприйняти	Продукт необхідний не всім	Висока конкуренція	Вхід не є складний
3	Державні ІТ-установи	Не всі користувачі готові сприйняти	Не всі зацікавлені в продукті	Середня конкуренція	Вхід складний
Які цільові групи обрано: Потрібно працювати зі всіма цільовими групами, перевагу надати приватними ІТ-компаніям та професійним користувачам, що займаються наданням послуг та розробкою ПЗ.					

Таблиця 4.14

Визначення базової стратегії розвитку

№	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Розвиток шляхом впровадження потужної рекламної кампанії	Реклама допоможе знайти нових користувачів та залучити інвесторів	Акцентувати увагу споживачів на функціональних можливостях та перспективі розвитку продукту	Оперативно оновлювати застосунок у відповідності до потреб клієнтів

Таблиця 4.15

Визначення базової стратегії конкурентної поведінки

№	Чи є проект «першо-проходцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1	Продукт є наступником вже існуючого на світовому рингу аналогу	Необхідно шукати нових споживачів та відбирати існуючих у конкурентів	Через схожість функціоналу, компанія буде копіювати основні характеристики у конкурентів	Пропонувати схожий функціонал на кращих умовах, оперативно оновлювати його

Таблиця 4.16

Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкуренто-спроможні позиції власного стартап- проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Зручність використання, надійність системи, постійні оновлення та підтримка додатку	Залучати все більше нових інвестицій для розширення, масштабування та підтримки застосунку	Оперативне виправлення недоліків та впровадження побажань	Зручний спосіб зберігання актуального стану портфоліо компанії, допомога в маркетинговій кампанії, залучення клієнтів

4.5. Розроблення маркетингової програми стартап-проекту

Таблиця 4.17

Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Збереження актуального стану портфоліо	Зручний та надійний менеджмент портфелю проектів	Автоматичне додавання чи оновлення одразу декількох проектів у портфоліо

Продовження таблиці 4.17

2	Часте оновлення застосунку	Врахування побажань користувачів та виправлення помилок в найкоротші терміни	Створення відділу підтримки для аналізу та оперативного реагування на відгуки клієнтів
3	Маркетинг власної компанії за рахунок налаштування портфоліо для специфічних клієнтів	Можливість створювати та поширювати лише окремі сети власних досліджень чи попередніх замовлень	Зручно налаштовувати на контролювати спеціалізовані набори робіт

Таблиця 4.18

Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові	
I. Товар за задумом	Система управління портфоліо ІТ-компанії для залучення нових клієнтів у вигляді веб-застосунку	
II. Товар у реальному виконанні	Властивості/характеристики	Значення
	1. Модуль для управління та поширення портфоліо	Основна система
	2. Модуль аутентифікації	Для авторизації користувачів
	3. Модуль відкладених задач	Аналіз результатів, повідомлення статистики

Продовження таблиці 4.18

	Якість: внутрішнє тестування програми, логування збоїв та система зворотного зв'язку для повідомлення про неналежну роботу додатку
	Пакування: веб-застосунок розміщений у мережі Інтернет
	Марка: «Менеджмент портфоліо»
III. Товар із підкріпленням	До продажу: стартап-ідея та програмний код
	Після продажу: повноцінна робоча система
За рахунок чого потенційний товар буде захищено від копіювання: здійснюється контрольований доступ до серверу, де знаходиться програмний код, надійно захищена база даних з конкретними ролями адміністрування, аутентифікація при кожній спробі доступу до середовища, сучасне шифрування каналів зв'язку	

Таблиця 4.19

Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі становлення ціни на товар/послугу
1	0\$ – 100\$	0\$ – 100\$	>1000\$/місяць	0\$ – 100\$

Таблиця 4.20

Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Безкоштовний доступ до застосунку, придбання преміум підписки, замовлення інтеграції системи	Створення та керування персональними даними, їх захист, підтримка проекту, консультація замовника	Канал збуту однорівневий (через хостинг на віддаленому сервері)	Вертикальна (право власності залишається у розробника ПЗ)

Таблиця 4.21

Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Доступ до застосунку за допомогою веб-браузера	Служба підтримки, електронна пошта,	Управління приватним та публічним портфоліо,	Демонстрація основних функцій розробленого	Охоплення аудиторії, пояснення функцій та

Продовження таблиці 4.21

		LinkedIn	розповсюдження окремих наборів серед спеціалізованих клієнтських аудиторій	продукту	можливостей розробленого продукту та його переваг
--	--	----------	---	----------	--

ВИСНОВКИ ДО РОЗДІЛУ 4

Четвертий розділ присвячений розробці стартап-проекту для системи управління та розповсюдження портфолію. Було зроблено опис ідеї проекту, проведено детальний аналіз конкурентів та можливостей, що доступні на даному ринку. Було наведено загальний опис ідеї проекту для виходу на ринок, описані можливості проекту, визначено основних конкурентів та співставлено можливості конкурентів для визначення переваг розроблюваного продукту над обраними конкурентами. Проведено технічний аудит проекту і визначено можливості реалізації програмного продукту. Було проаналізовано ринкові можливості для запуску проекту, порівняно перспективи запуску в порівнянні з конкурентами, встановлено план та ринкову стратегію розвитку продукту, визначено цільові групи та способи просування проекту. Розроблено маркетингову програму для просування продукту на ринку, описано способи та основні канали збуту, визначено пріоритетні цільові групи та маркетингові повідомлення для розширення клієнтської бази.

В цілому, було розроблено повноцінний стартап-проект для запуску розробленого програмного продукту на ринок, отримано навички створення стартап-проектів та побудови маркетингової стратегії, аналізу обраного ринку.

ВИСНОВКИ

У результаті роботи було проаналізовано вимоги та розроблено модульну систему, що здатна до масштабування, є функціональною та економічно конкурентоспроможною.

У цій магістерській роботі було розглянуто проблематику менеджменту портфолію власних проектів з можливостями подальшого розповсюдження своїх досліджень для зацікавлення нових клієнтів та залучення інвестицій. Створений застосунок дозволяє тримати портфолію в актуальному стані, а також вирішує проблему маркетингу власного бренду та збагачення клієнтської аудиторії.

У першому розділі було проаналізовано існуючі рішення та встановлено їх ключові переваги й недоліки. Згідно з проведеним аналізом було визначено головні критерії для розробки та актуальність проблематики.

Розглянувши основні вимоги до системи, у другому розділі було проведено дослідження та визначено головні технології для розробки застосунку. Згідно з загальновідомими стандартами побудови архітектури програмного забезпечення, було використано шаблони проектування, що надають системі здатність до гнучкого масштабування та оновлення. Основними технологіями було визначено: мову програмування Python і фреймворк Django для розробки серверної частини, та мову JavaScript у поєднанні з фреймворками Bootstrap та JQuery для реалізації клієнтської частини.

У третьому розділі було повністю описано розробку бізнес-логіки системи, а також інтерфейсу користувача. Також було інтегровано додаткові сервіси для високопродуктивного повнотекстового пошуку та виконання відкладених задач. Для моніторингу цих процесів та сповіщення про помилки й несправності системи використовувався зручний графічний інтерфейс окремого інструменту.

У четвертому розділі систему було розглянуто як стартап-проект з можливістю виходу на ринок. Було виконано опис ідеї, аналіз сильних та слабких сторін проекту, потенційних можливостей росту, а також загроз для подальшого

розвитку. Було проведено аналіз ринкової можливості для запуску проекту та визначено цільові групи серед яких даний проект буде просуватися.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What exactly is a marketing portfolio and why do you need one? [Електронний ресурс]. — Режим доступу: <https://careerfoundry.com/en/blog/digital-marketing/create-marketing-portfolio/> — Дата доступу : Вересень 2022.
2. The best digital marketing portfolio of 2022. [Електронний ресурс]. — Режим доступу: <https://www.twine.net/blog/best-digital-marketing-portfolio/> — Дата доступу : Вересень 2022.
3. Portfolio Management. [Електронний ресурс]. — Режим доступу: <https://www.pmi.org/learning/featured-topics/portfolio> — Дата доступу : Вересень 2022.
4. The opportunities of Coroflot. [Електронний ресурс]. — Режим доступу: <https://www.coroflot.com/about> — Дата доступу : Вересень 2022.
5. Flickr Pro. [Електронний ресурс]. — Режим доступу: <https://www.flickr.com/account/upgrade/pro> — Дата доступу : Вересень 2022.
6. Entsane: entertainment industry talent and freelancers. [Електронний ресурс]. — Режим доступу: <https://www.entsane.org/about-us/> — Дата доступу : Вересень 2022.
7. About Crevado. [Електронний ресурс]. — Режим доступу: <https://crevado.com/about> — Дата доступу : Жовтень 2022.
8. PYPL Popularity of Programming Language. [Електронний ресурс]. — Режим доступу: <https://pypl.github.io/PYPL.html> — Дата доступу : Жовтень 2022.
9. How does Python work? [Електронний ресурс]. — Режим доступу: <https://towardsdatascience.com/how-does-python-work-6f21fd197888> — Дата доступу : Жовтень 2022.
10. Mark Lutz, Learning Python. O'Reilly. 2011. 720 с.

11. Mark Lutz, Learning Python: Powerful Object-Oriented Programming, vol. 1. O'Reilly. – 2011. 992 с.
12. PyPi: The Python Package Index. [Електронний ресурс]. — Режим доступу: <https://pypi.org/> — Дата доступу : Жовтень 2022.
13. JetBrains Statistics: Python Usage. [Електронний ресурс]. — Режим доступу: <https://www.jetbrains.com/lp/devecosystem-2020/python/> — Дата доступу : Листопад 2022.
14. What is Java? [Електронний ресурс]. — Режим доступу: <https://aws.amazon.com/what-is/java/> — Дата доступу : Листопад 2022.
15. Definition, Meanings & Features of Java Platforms. [Електронний ресурс]. — Режим доступу: <https://www.guru99.com/java-platform.html> — Дата доступу : Листопад 2022.
16. Understanding Java Virtual Machine Architecture. [Електронний ресурс]. — Режим доступу: <https://medium.com/java-for-beginners/understanding-java-virtual-machine-jvm-architecture-e68d1c611026> — Дата доступу : Листопад 2022.
17. Understanding JVM Architecture. [Електронний ресурс]. — Режим доступу: <https://medium.com/platform-engineer/understanding-jvm-architecture-22c0dddf09722> — Дата доступу : Листопад 2022.
18. JetBrains Statistics: Java Usage. [Електронний ресурс]. — Режим доступу: <https://www.jetbrains.com/lp/devecosystem-2020/java/> — Дата доступу : Листопад 2022.
19. Django Documentation. [Електронний ресурс]. — Режим доступу: <https://docs.djangoproject.com/en/4.1/> — Дата доступу : Листопад 2022.
20. PostgreSQL Documentation. [Електронний ресурс]. — Режим доступу: <https://www.postgresql.org/docs/> — Дата доступу : Листопад 2022.

- 21 PostgreSQL vs MySQL: Explore Their 12 Critical Differences. [Електронний ресурс]. — Режим доступу: <https://kinsta.com/blog/postgresql-vs-mysql/> — Дата доступу : Листопад 2022.
- 22 PostgreSQL vs MySQL: What's the difference? [Електронний ресурс]. — Режим доступу: <https://www.ibm.com/cloud/blog/postgresql-vs-mysql-whats-the-difference> — Дата доступу : Листопад 2022.
- 23 PostgreSQL vs MySQL. [Електронний ресурс]. — Режим доступу: <https://www.sumologic.com/blog/postgresql-vs-mysql/> — Дата доступу : Листопад 2022.
- 24 Celery Tutorial: A Must-Learn Technology for Python Developers. [Електронний ресурс]. — Режим доступу: <https://medium.com/swlh/python-developers-celery-is-a-must-learn-technology-heres-how-to-get-started-578f5d63fab3> — Дата доступу : Листопад 2022.
- 25 HTML Introduction. [Електронний ресурс]. — Режим доступу: https://www.w3schools.com/html/html_intro.asp — Дата доступу : Листопад 2022.
- 26 CSS Tutorial. [Електронний ресурс]. — Режим доступу: <https://www.w3schools.com/css/default.asp> — Дата доступу : Листопад 2022.
- 27 Introduction to Bootstrap 4. [Електронний ресурс]. — Режим доступу: <https://getbootstrap.com/docs/4.6/getting-started/introduction/> — Дата доступу : Листопад 2022.
- 28 The Django template language. [Електронний ресурс]. — Режим доступу: <https://docs.djangoproject.com/en/4.1/ref/templates/language/> — Дата доступу : Листопад 2022.
- 29 JavaScript. [Електронний ресурс]. — Режим доступу: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> — Дата доступу : Листопад 2022.

30. What is JQuery? [Электронный ресурс]. — Режим доступа:
<https://jquery.com/> — Дата доступа : Листопад 2022.
31. What is ElasticSearch. [Электронный ресурс]. — Режим доступа:
<https://www.elastic.co/what-is/elasticsearch> — Дата доступа : Грудень 2022.
32. Flower – Celery monitoring tool. [Электронный ресурс]. — Режим доступа:
<https://flower.readthedocs.io/en/latest/> — Дата доступа : Грудень 2022.

Структурна схема роботи застосунку

