ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення для аналізу вакансій з використанням фреймворку Spring Boot, мовою Java»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Андрій КОНОПЛЯСТИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Андрій КОНОПЛЯСТИЙ

Керівник: Вікторія ЖЕБКА
д.т.н., професор

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Коноплястому Андрію Руслановичу

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення для аналізу вакансій з використанням фреймворку Spring Boot, мовою Java»
керівник кваліфікаційної роботи д.т.н., професор, зав. кафедри ТЦР Вікторія ЖЕБКА,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи:
   1. Науково-технічна література, пов’язана із розробкою веб-додатків
   2. Мова програмування Java
   3. Мова програмування JavaSript
   4. Фреймворк Angular
   5. Фреймворк для тестування Mockito та JUnit
   6. Система управління базами даних PostgreSQL

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   1. Аналіз предметної області

2. Аналіз та дослідження існуючих аналогів
3. Технічне завдання
4. Моделювання архітектури системи
5. Розробка програмного забезпечення
6. Тестування розробленого програмного забезпечення
7. Висновки

5. Перелік графічного матеріалу: *презентація*
1. Мета, об’єкт, предмет, наукова новизна дослідження.
2. Задачі дипломної роботи.
3. Аналіз аналогів.
4. Вимоги до програмного забезпечення
5. Програмні засоби реалізації
6. Структура бази даних
7. Діаграма класів
8. Екранні форми
9. Апробація результатів дослідження

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Проектування системи	14.03-18.03.2024	
4	Оптимізація та налаштування системи	19.03-31.03.2024	
5	Програмна реалізація застосунку	1.04-18.04.2024	
6	Тестування застосунку	20.04-25.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Андрій КОПОПЛЯСТИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Вікторія ЖЕБКА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 48 стор., 1 табл., 26 рис., 14 джерел.

Мета роботи – спрощення процесу аналізу вакансій за допомогою веб-застосунку, який використовує мову Java та фреймворки Spring Boot і Angular.

Об'єкт дослідження – аналіз вакансій та ринку праці в IT.

Предмет дослідження – програмне забезпечення для аналізу, пошуку, та візуалізації вакансій, включаючи алгоритми обробки даних та інтерфейси користувача.

Короткий зміст роботи: В роботі описано підходи до створення веб-застосунку, який спрощує пошук вакансій у сфері IT. Заплановано інтеграцію алгоритмів штучного інтелекту для аналізу вакансій і прогнозування технологічних тенденцій, яка буде впроваджена у майбутньому. Веб-застосунок дозволяє користувачам отримувати актуальні вакансії відповідно до поточних та майбутніх потреб ринку праці. Розроблено застосунок використовуючи Java та Spring для серверної частини та Angular для фронтенду, забезпечуючи таким чином високу продуктивність та адаптивність інтерфейсу. У роботі використано сучасні технології та методики розробки, що дозволяє легко адаптувати систему під майбутні розширення функціоналу.

Сферою використання застосунку є аналіз та пошук IT-вакансій для оптимізації процесу знаходження роботи.

КЛЮЧОВІ СЛОВА: Full stack, Java, JavaScript, Spring Boot, Angular, RESTful API.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. HTML – HyperText Markup Language
2. CSS – Cascading Style Sheets
3. SCSS – Sassy
4. JS – JavaScript
5. TS – TypeScript
6. NG – Angular
7. IDE – Integrated Development Environment
8. API – Application Programming Interface
9. REST – Representational State Transfer
10. HTTP – HyperText Transfer Protocol
11. JPA – Java Persistence API
12. API SQL – Structured Query Language
13. СУБД – Система управління базами даних
14. ОС – Операційна система
15. CI/CD – Continuous Integration/Continuous Deployment
16. XML – Extensible Markup Language
17. JSON – JavaScript Object Notation
18. YAML – YAML Ain’t Markup Language
19. JAR – Java Archive
20. DTO – Data Transfer Object
21. REST – Representational State Transfer
22. UUID – Universally Unique Identifier
23. UML – Unified Modeling Language
24. ПЗ – Програмне забезпечення

ВСТУП

Обґрунтування вибору теми та її актуальність: В умовах стрімкого розвитку інформаційних технологій та постійного збільшення попиту на кваліфіковані кадри, актуальність створення надійних та ефективних інструментів для пошуку роботи в IT галузі зростає. Цей проект має на меті розв'язати проблему нестачі спеціалізованих інструментів для аналізу та відстеження кар'єрних можливостей у цій швидкозмінній сфері.

Об’єктом дослідження є процес аналізу та пошуку вакансій ринку праці в IT-галузі.

Предмет дослідження веб-застосунок, розроблений для ефективного пошуку вакансій та аналізу ринку праці в IT.

Метою роботи є розробка веб-застосунку, який дозволить користувачам здійснювати швидкий та цілеспрямований пошук вакансій, а також аналізувати поточні тренди та вимоги ринку праці в галузі інформаційних технологій.

Методи дослідження: включають використання мови програмування Java для бекенду, фреймворку Spring для серверної частини, фреймворку Angular для фронтенду, а також аналіз даних для оцінки потреб ринку.

Наукова новизна роботи полягає у створенні інтегрованого інструменту, який комбінує в собі можливості пошуку роботи та аналізу ринку праці, забезпечуючи користувачам доступ до актуальних даних про вакансії та тенденції у галузі в реальному часі.

Практична значущість результатів дослідження полягає в створенні застосунку, який не тільки підвищує шанси IT-спеціалістів знайти відповідну роботу, але й дозволяє компаніям ефективніше знаходити кваліфікованих працівників. Веб-застосунок сприяє стратегічному плануванню кар'єри та підвищує залученість користувачів у процес пошуку роботи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд ринку праці в IT-галузі

Інформаційні технології є однією з найшвидше розвиваючихся галузей сучасності, і цей процес не зупиняється ні на хвилину. Завдяки невпинному прогресу технологій, з'являються нові можливості для професійного зростання та інновацій. Незмінним залишається високий попит на IT-спеціалістів, що стимулює зростання кількості вакансій в цьому секторі по всьому світу.

Ринок праці в IT відрізняється високою динамікою. Технологічні компанії, стартапи, а також традиційні бізнеси, що активно інтегрують цифрові рішення, постійно шукають кваліфікованих розробників, дата-аналітиків, системних архітекторів та інших фахівців. Цікаво, що зростання потреби в робочій силі спостерігається не тільки в крупних технологічних центрах, як-от Силіконова долина чи Бангалор, але й у більш малих містах та регіонах, де технологічні компанії відкривають свої офіси з метою зниження витрат та доступу до нових талантів.

Однак не можна ігнорувати вплив глобальних подій на IT-галузь. Наприклад, нещодавні військові конфлікти, такі як війна в Україні, внесли свої корективи в роботу багатьох IT-компаній. Війна не тільки спровокувала переосмислення стратегій безпеки і резервування даних, але й викликала хвилю міграції IT-спеціалістів, що в свою чергу вплинуло на ринки праці країн-сусідів. Це, з одного боку, збільшило пропозицію кваліфікованих фахівців в певних регіонах, а з іншого – поставило питання адаптації та інтеграції цих спеціалістів в нові робочі культури.

На ринку праці в IT існує також значний інтерес до розробників із вміннями в новітніх технологіях, таких як штучний інтелект, машинне навчання, великі дані, а також фахівців у сфері кібербезпеки. Все більше компаній оцінюють кіберзахист як один з пріоритетів своєї діяльності, особливо

в контексті збільшення кількості кібератак та потреби у захисті корпоративних даних.

Крім того, тенденції на ринку праці в IT вказують на зростання попиту на розробників у сфері хмарних технологій та інтеграції систем. Це пов'язано з продовженням трансформації бізнесів у напрямку цифровізації, де важливою стає можливість швидкої адаптації до змінюваних умов ринку і нових технологічних рішень. Також значну роль відіграє збільшення кількості стартапів, які намагаються впровадити інноваційні рішення, що часто потребують спеціалістів унікальних профілів та високого рівня кваліфікації.

Огляд ринку праці в IT показує, що незважаючи на виклики, що виникають через глобальні події або економічні коливання, сфера інформаційних технологій залишається однією з найбільш стабільних і високооплачуваних. Важливо зазначити, що динаміка ринку праці в IT також спонукає спеціалістів до постійного навчання та професійного розвитку, щоб бути в курсі новітніх технологічних трендів та відповідати вимогам роботодавців. Це створює попит на освітні програми та курси, які можуть підготувати IT-фахівців до вирішення складних завдань сучасного технологічного світу.

Загалом, аналіз ринку праці в IT є важливим інструментом для розуміння поточних і майбутніх потреб галузі, і цей огляд є частиною підготовки до створення веб-застосунку, який зможе забезпечити ефективний зв'язок між роботодавцями та потенційними працівниками, сприяючи позитивним змінам на ринку праці.

1.2 Огляд та аналіз існуючих аналогів

Під час аналізу існуючих аналогів веб-застосунків для пошуку роботи в IT, особливу увагу слід приділити платформам, які вже зарекомендували себе на ринку Dou та Djinni. Таблиця порівняння (див. табл. 1), демонструє, які

функції присутні (позначено "+") та відсутні (позначено "-") у нашому застосунку порівняно з аналогами.

Таблиця 1.1

Порівняння аналогів з розробленим додатком

Функціонал	Djinni	Dou.ua	VA
Відображення вакансій з інших платформ	-	-	+
Розширений пошук за категоріями та фільтрами	+	+	+
Розділ блогів і статей	-	+	+
Статистика зарплат	+	+	+
Широкий спектр вакансій у різних галузях	-	-	+
Прямий контакт з роботодавцями	+	-	+

1.2.1 Dou

Dou — це відома українська платформа (див. рис. 1.1), яка спеціалізується на забезпеченні IT-спеціалістів вакансіями від провідних технологічних компаній. Особливістю Dou є її спільнота: портал активно використовується професіоналами для обміну досвідом та знаннями через статті, обговорення та огляди (див. рис. 1.2).

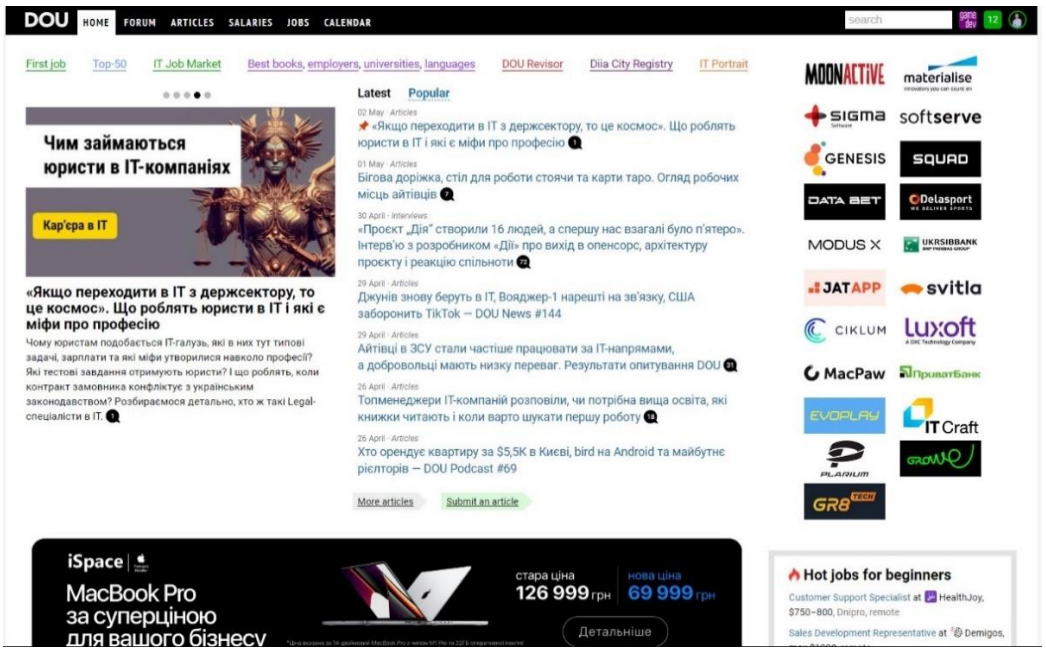


Рис. 1.1 Головна сторінка веб-платформи Dou

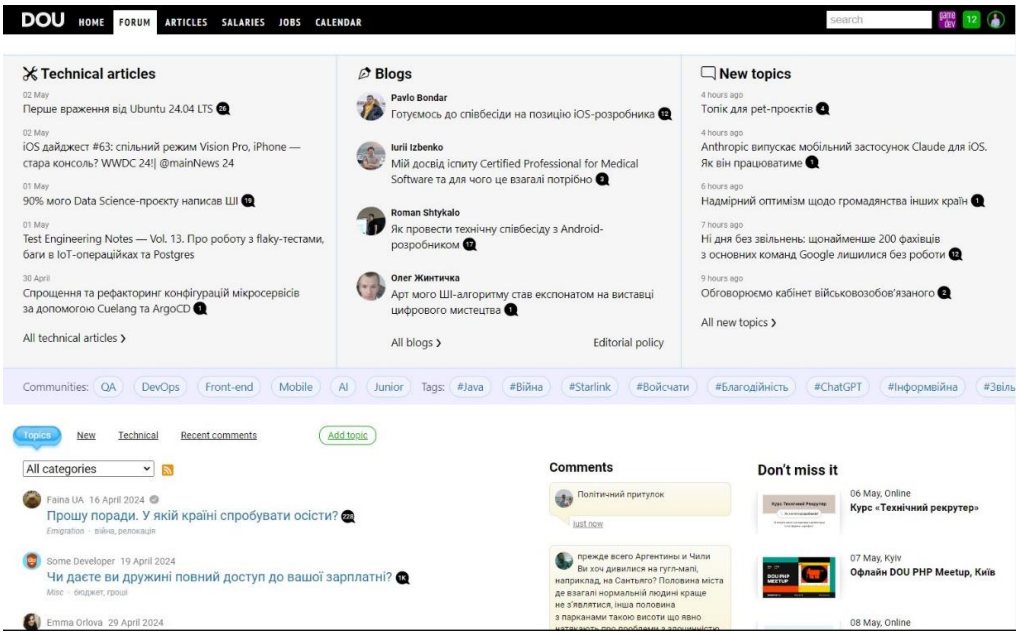


Рис. 1.2 Сторінка спілкування з іншими користувачами

Головні функції:

– Розширений пошук вакансій за ключовими словами, технологіями, локацією, і рівнем зарплати;

– Огляди зарплат, умов праці в різних компаніях;
– Щомісячні аналітичні звіти про стан IT-ринку;
– Спільнота з можливістю обговорення актуальних тем, створення контенту користувачами;

Переваги:

- Значна кількість користувачів, що складається з активних IT-спеціалістів;
- Широкий спектр інформації про робочі умови і корпоративну культуру компаній;
- Велика кількість унікального контенту, спрямованого на розвиток кар'єри;
- Можливість відгукнутися на вакансію без створення аккаунта на платформі;

Недоліки:

- Відсутність інтеграції з іншими міжнародними платформами пошуку роботи.
- Обмежені можливості персоналізації досвіду користувача на сайті.
- Основний фокус на українському ринку, що може обмежувати міжнародних користувачів.

1.2.2 Djinni

Djinni — це інноваційна платформа для пошуку роботи в IT, яка забезпечує анонімність кандидатів до моменту виявлення зацікавленості з боку роботодавця (див. рис. 1.3 – 1.4). Це дозволяє фахівцям вільно розміщувати свої резюме, не побоюючись непередбачених реакцій поточних роботодавців.

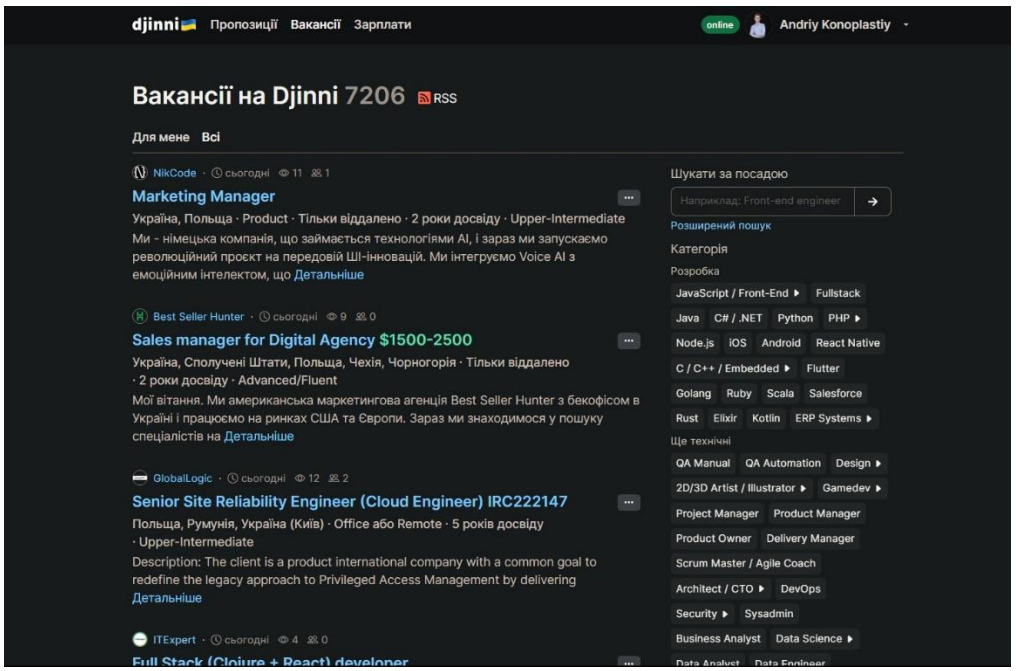


Рис. 1.3 Головна сторінка веб-платформи Djinni

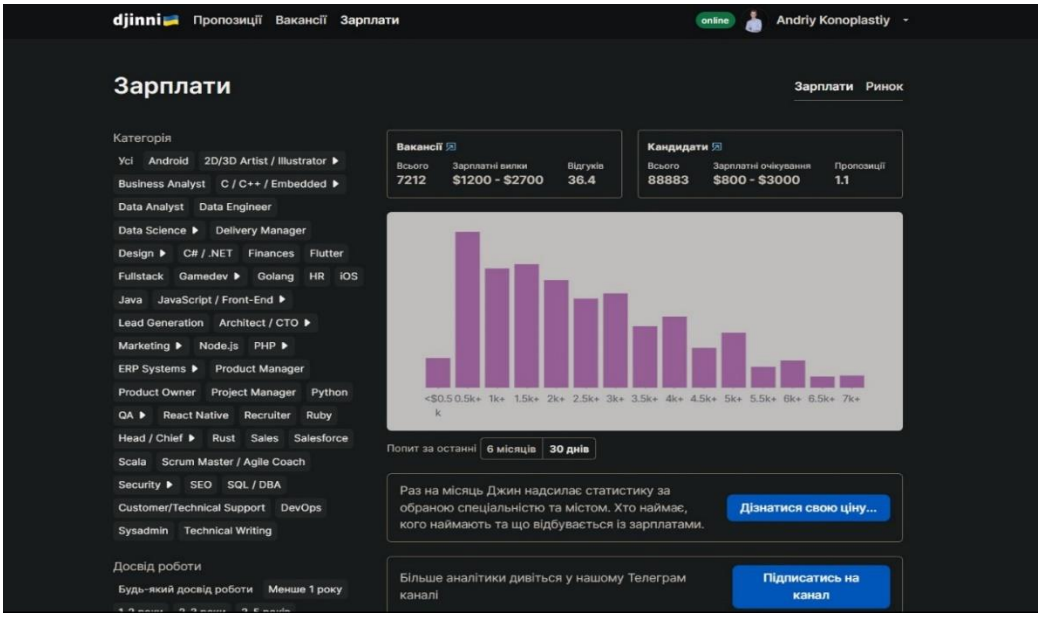


Рис. 1.4 Сторінка зі статистикою та аналізом рівня зарплат на платформі Djinni

Головні функції:

- Анонімне розміщення резюме кандидатами;
- Прямий зв'язок між кандидатами та роботодавцями без посередників;

– Автоматичне сповіщення кандидатів про нові вакансії, що відповідають їхнім критеріям;

Переваги:

- Висока конфіденційність для шукачів, що сприяє збільшенню кількості якісних резюме;
- Інтеграція з LinkedIn для перевірки професійних даних;
- Простота використання і ефективність пошуку завдяки прямому зв'язку між сторонами;
- Активна підтримка з боку адміністрації сайту у вирішенні спірних питань;

Недоліки:

- Анонімність може ускладнити вибір кандидатів для деяких роботодавців;
- Обмежені можливості для маркетингу власного бренду кандидатом;
- Необхідність активної участі кандидата у процесі пошуку може бути викликом для зайнятих фахівців;
- Відгук на вакансію можливий тільки після отримання відповіді від HR, який її опублікував;

2 ЗАСОБИ РОЗРОБКИ

2.1 Середовище розробки та засоби тестування

IntelliJ IDEA – розроблена компанією JetBrains, є одним із найвідоміших інтегрованих середовищ розробки (IDE) для Java. Це середовище славиться своєю адаптивністю та розширеним набором можливостей, які включають підтримку різноманітних мов програмування, зокрема Kotlin , SQL, Scala, JavaScript, TypeScript та інші мови програмування. IntelliJ IDEA пропонується у двох варіантах: безкоштовний Community Edition та платний Ultimate Edition. Розглянемо переваги та недоліки обох цих версій (див. рис. 2.1 – 2.3).

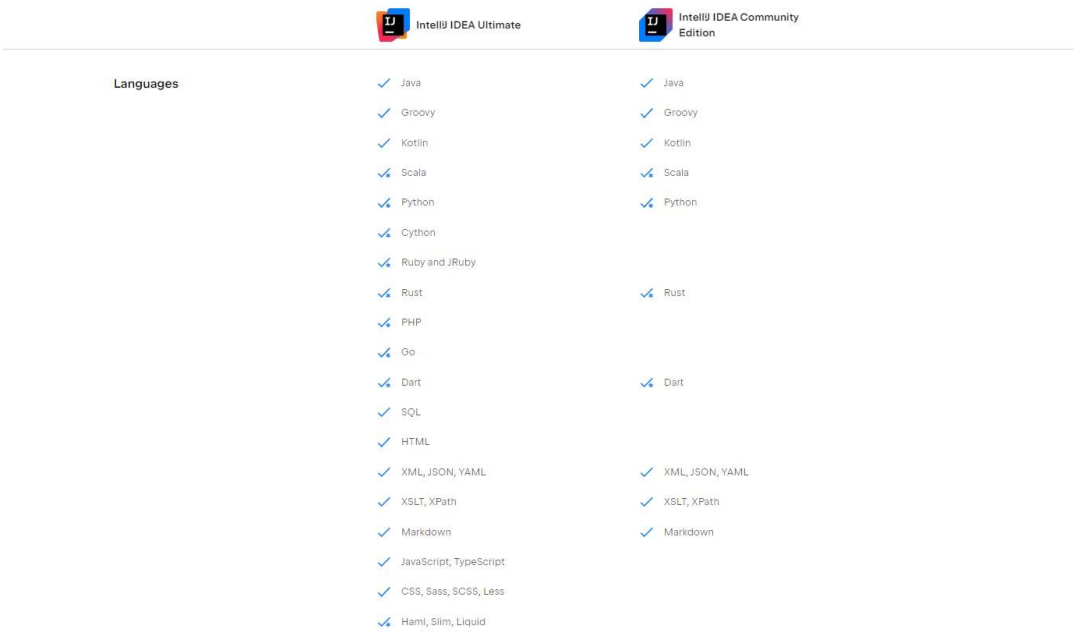


Рис. 2.1 Порівняння підтримки мов програмування в Community і Ultimate версіях IntelliJ IDEA.

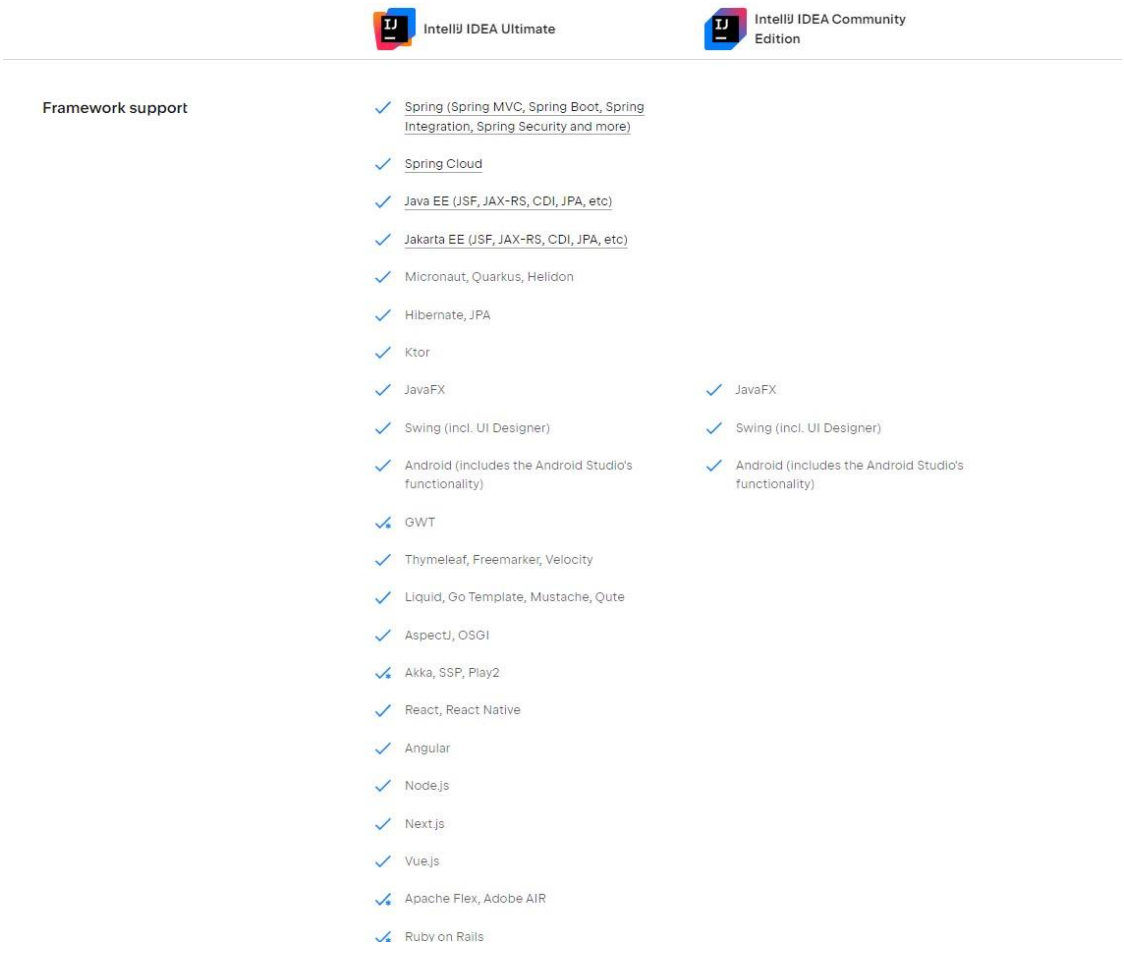


Рис. 2.2 Порівняння підтримки фреймворків у Community і Ultimate версіях IntelliJ IDEA.

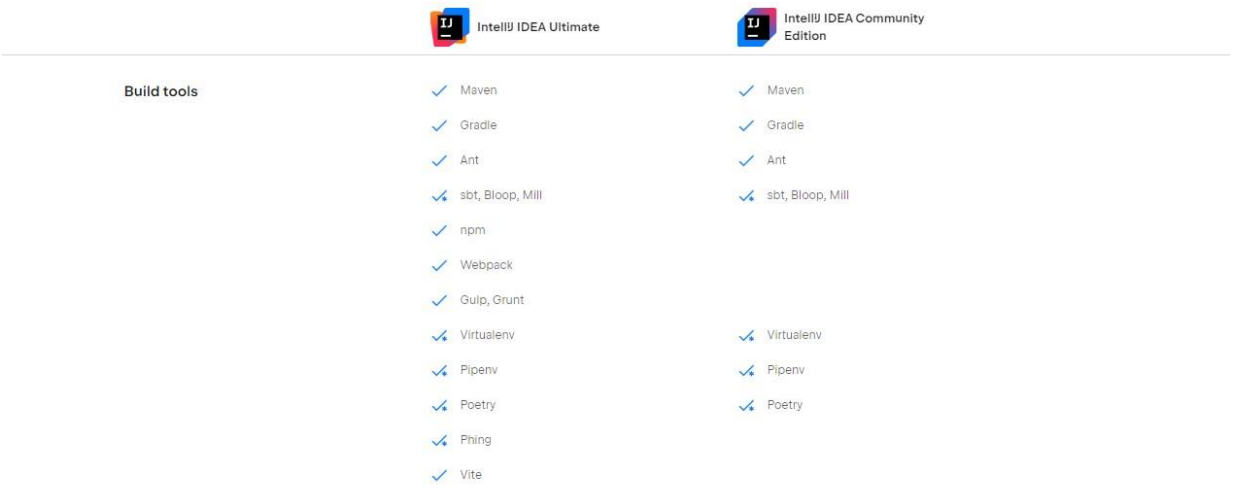


Рис. 2.3 Порівняння доступності додаткових інструментів у Community і Ultimate версіях IntelliJ IDEA.

Postman – це потужний інструмент для розробки API (див. рис. 2.4), який дозволяє розробникам легко створювати, тестувати, документувати та спільно використовувати API. Він пропонує інтуїтивно зрозумілий інтерфейс для відправки запитів, перевірки відповідей, а також підтримує автоматизацію тестування. За допомогою колекцій і середовищ, Postman спрощує спільну роботу та інтеграцію різних API.

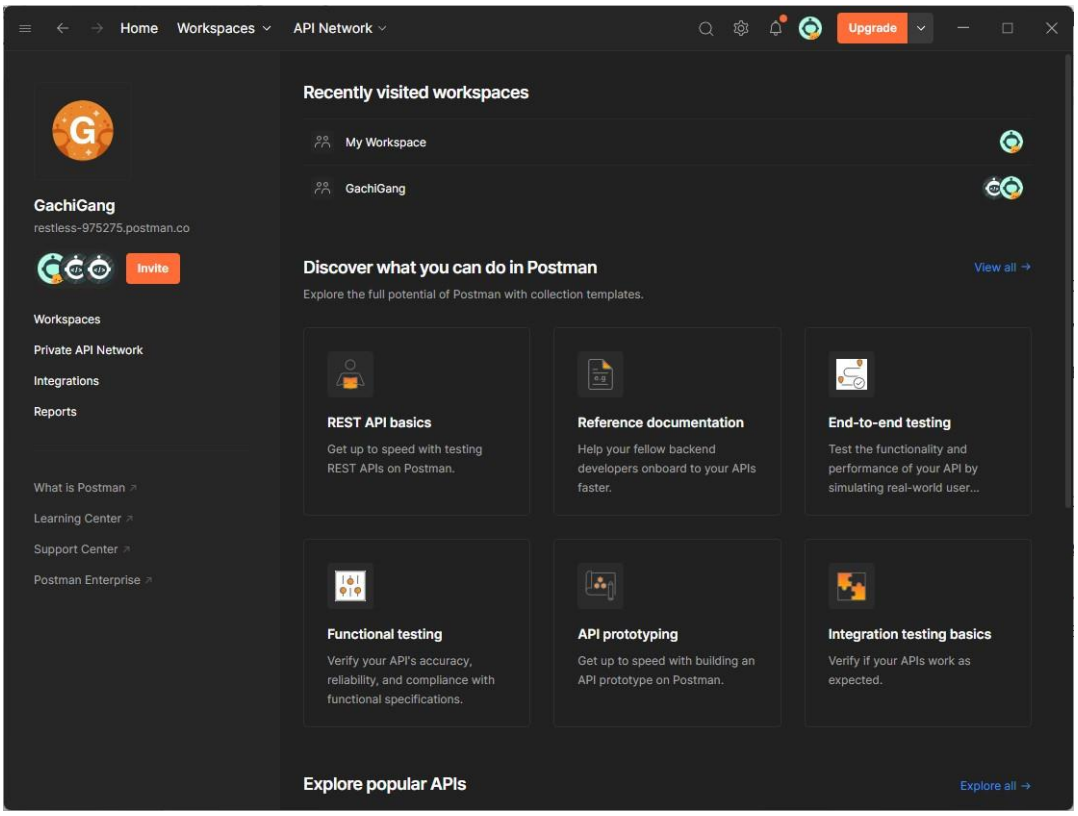


Рис. 2.4 Скріншот головної сторінки інтерфейсу застосунку Postman

Selenium – це інструмент (див. рис. 2.5), який часто використовується не тільки для тестування веб-інтерфейсів, але й для автоматизації завдань веб-парсингу. Завдяки своїй здатності імітувати реальні дії користувача в браузері, Selenium дозволяє ефективно витягувати дані з динамічних веб-сторінок, які змінюють вміст у відповідь на дії користувача, забезпечуючи доступ до інформації, яка не доступна через стандартні методи HTTP-запитів.

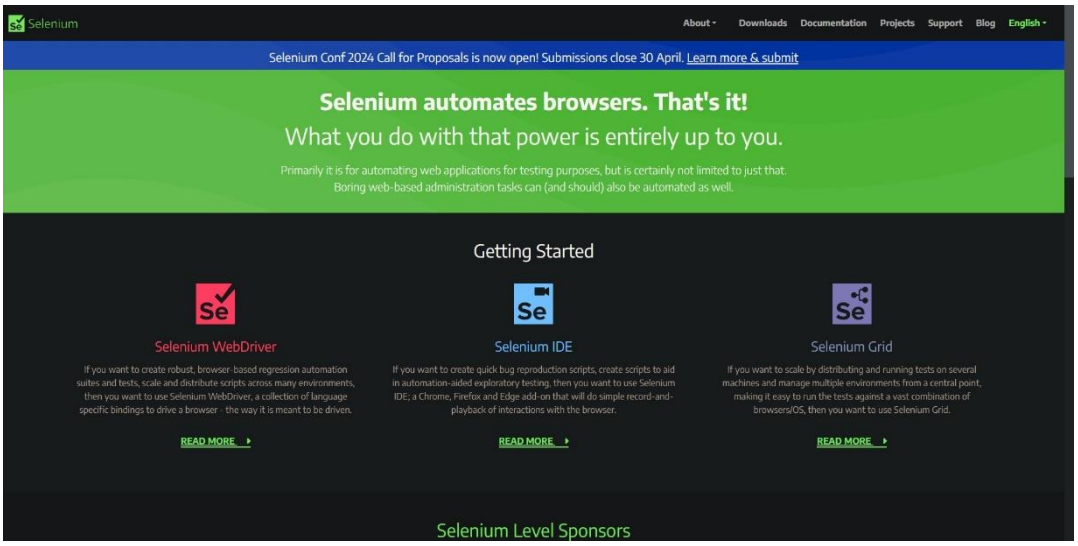


Рис. 2.5 Головна сторінка веб-платформи Selenium

2.2 Мови програмування

Java – це високорівнева, класно-орієнтована, об'єктно-орієнтована мова програмування, яка була розроблена компанією Sun Microsystems в середині 90-х років. Завдяки своїй архітектурній незалежності Java має велику популярність у розробці крос-платформних додатків. Вона широко використовується для створення веб-серверних додатків, корпоративного програмного забезпечення та мобільних додатків на платформі Android. Мова славиться своєю надійністю, масштабованістю та багатопоточністю. Java вимагає компіляції в байт-код, який виконується на Java Virtual Machine (JVM), забезпечуючи високу безпеку та переносимість коду.

Особливості JavaScript включають:

- Об'єктно-орієнтованість, що сприяє чистоті коду та перевикористанню компонентів.
- Незалежність від платформи, дозволяючи програмам працювати на будь-якій системі з встановленою JVM.

- Велика кількість бібліотек і фреймворків, таких як Spring та Hibernate, які спрощують розробку складних застосунків.
- Сильна спільнота та підтримка, включаючи обширну документацію та ресурси для навчання.

Недоліки JavaScript:

- Велика споживання пам'яті, особливо в додатках, що інтенсивно використовують об'єкти.
- Відносно повільна швидкість виконання порівняно з компільованими мовами, такими як C++.
- Затримка у збиранні сміття, яка може призвести до періодичних заминок у виконанні програми, особливо в системах з великою кількістю об'єктів у пам'яті.
- Залежність від зовнішнього середовища виконання (JVM), що іноді може створювати проблеми з сумісністю на різних платформах, незважаючи на принцип "Write Once, Run Anywhere".
- Не завжди ідеально підходить для сценаріїв, що вимагають високої продуктивності в обчисленнях, таких як ігри або наукові додатки, де критично важлива швидкість виконання.

JavaScript — це інтерпретована мова програмування, що є необхідною для веб-розробки. Вона використовується для створення інтерактивних веб-сторінок і є майже універсальним стандартом для клієнтських скриптів на веб-сайтах. JavaScript дозволяє динамічно змінювати вміст сторінки, стилі, і взаємодіяти з браузером, що робить веб-досвід більш багатим та взаємодійним. Ця мова також може використовуватися на серверній стороні (Node.js), що робить її ще більш потужною та універсальною для повного стеку веб-розробки. Незважаючи на свою простоту і гнучкість, JavaScript часто критикують за неоднозначність та неконсистентність деяких своїх аспектів, що може призвести до помилок при розробці.

Особливості JavaScript включають:

- Виконання на клієнтській стороні для негайної взаємодії з користувачем без потреби перезавантаження сторінки.
- Підтримка всіх сучасних браузерів, що робить JavaScript майже універсальним інструментом для веб-розробки.
- Можливість використання на серверній стороні через Node.js, що дозволяє розробникам використовувати одну мову на всіх етапах розробки.
- Велика кількість фреймворків та бібліотек, таких як React, Angular, та Vue.js, які полегшують створення інтерактивних веб-додатків.

Недоліки JavaScript:

- Часто критикується за неконсистентність та високу складність освоєння через неоднозначності у синтаксисі та поведінці.
- Відсутня строга типізація даних.
- Не реалізовані принципи о’єктно-оріентованого програмування.
- Можливості мови досить обмежені, для їх розширення необхідно підключати або створювати сторонні бібліотеки та фреймворки.

2.3 Фреймворки

Фреймворк у контексті програмування — це структурована платформа, яка слугує як фундамент для розробки додатків. Фреймворки надають готові компоненти та шаблони, що дозволяють розробникам швидше та ефективніше створювати додатки, зосереджуючись на бізнес-логіці, а не на базовій інфраструктурі. Вони також сприяють стандартизації коду та можуть значно знизити кількість помилок завдяки використанню перевірених і оптимізованих компонентів.

2.3.1 Spring Framework

Spring Framework - це вражаючий інструмент для розробки програмного забезпечення на Java. Він дозволяє розробникам створювати різноманітні застосунки з великою легкістю та ефективністю. Розроблений у 2003 році Родом Джонсоном, Spring швидко став відомим завдяки своїм унікальним можливостям та простоті використання.

Однією з ключових функцій Spring є концепція інверсії контролю (IoC) та аспектно-орієнтованого програмування (AOP), які допомагають писати більш зрозумілий та організований код. Фреймворк також має свій власний контейнер, який керує компонентами додатку та їхніми життєвими циклами.

Spring є відмінним вибором для будь-якого проекту - від найпростіших веб-сайтів до складних корпоративних застосунків. Він підтримує транзакції, має широкі можливості інтеграції та активно розвивається завдяки захоплюючій спільноті користувачів та розробників.

Мета створення Spring полягала у спрощенні розробки програмного забезпечення, роблячи його більш модульним та ефективним. Фреймворк постійно адаптується до нових викликів у технологічній сфері, залишаючись одним з найпопулярніших інструментів для розробників Java-додатків.

Основні переваги Spring Framework включають:

- Інверсія контролю та аспектно-орієнтоване програмування: Забезпечує організований та зрозумілий код.
- Модульність: Робить розробку, тестування та супровід коду більш ефективними.
- Розширюваність: Дозволяє інтегрувати різноманітні розширення та інші технології.
- Підтримка транзакцій: Ефективне управління транзакціями баз даних та іншими ресурсами.
- Широка спільнота: Активна спільнота користувачів та розробників, яка надає підтримку та рішення на будь-які питання.

- Інтеграція з іншими технологіями: Працює з іншими популярними технологіями.
- Спрощення тестування: Забезпечує зручні інструменти для написання автоматизованих тестів.

Для створення проекту з використанням Spring можна скористатися веб-сайтом https://start.spring.io/ (див. рис. 2.6). Цей інструмент надає зручний і простий спосіб створення проекту Spring Boot з необхідними залежностями та конфігураціями. Користувач може обирати потрібні технології, версії та інші параметри, і після цього автоматично згенерувати початковий код проекту. Це дозволяє швидко почати роботу над проектом без необхідності вручну налаштовувати всю конфігурацію. Такий підхід спрощує процес розробки та дозволяє швидше перейти до написання бізнес-логіки вашого додатку.

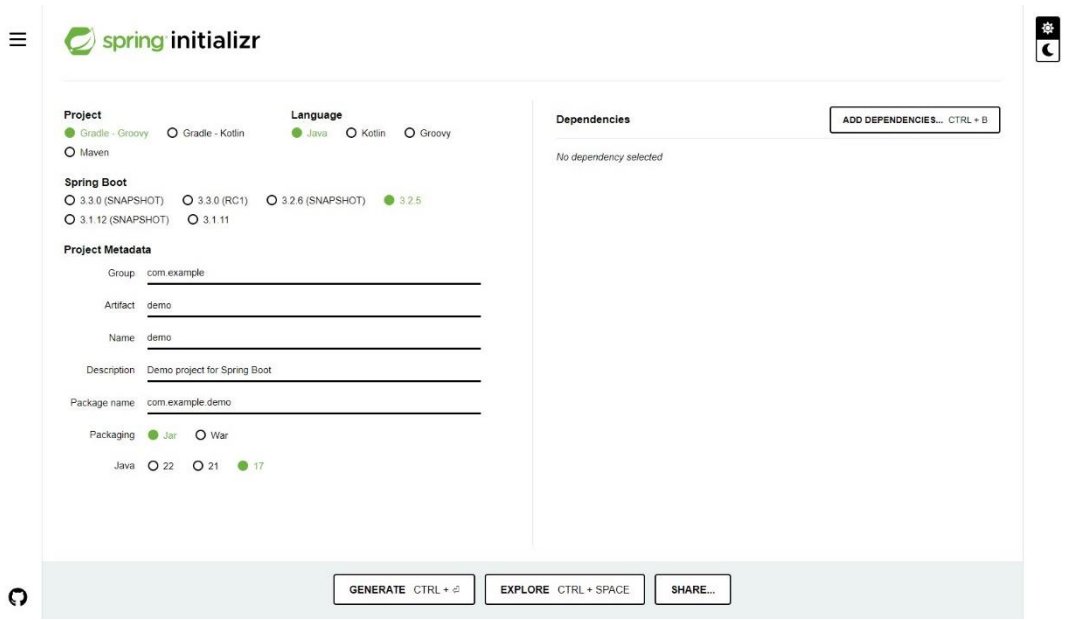


Рис. 2.6 Головна сторінка веб-додатку Spring Initializr

Spring Boot - це інструмент для швидкої розробки програм на Java, що надає широкі можливості без головних болів, пов'язаних з налаштуванням і конфігурацією. Він стає дуже привабливим для розробників завдяки своїм готовим шаблонам і конфігураціям, які значно спрощують створення нових проектів. За допомогою Spring Boot можна швидко розпочати роботу над проектом, не витрачаючи часу на налаштування з нуля.

Однією з ключових переваг Spring Boot є його вбудована підтримка вбудованих серверів, що дозволяє легко запускати додатки безпосередньо з вашої IDE або терміналу. Такий підхід зекономить ваш час, оскільки ви можете уникнути витрат на налаштування сервера.

Щодо використання, багато великих та відомих компаній, які використовують Spring у своїх проектах:

- Netflix
- LinkedIn
- Alibaba
- eBay
- Cisco
- Capital
- Uber
- Deliveroo
- SoundCloud

Це свідчить про популярність та довіру до цього фреймворку в розробці програмного забезпечення.

2.3.2 Angular Framework

Angular є одним із найпопулярніших фреймворків для розробки веб-додатків. Створений і підтримуваний Google, він з’явився у 2010 році як AngularJS, що використовував JavaScript. Проте, починаючи з 2016 року, Angular перейшов на мову TypeScript, що забезпечує більш строгу типізацію і об'єктно-орієнтовані можливості, що робить його ідеальним для розробки великих додатків.

Angular має багато особливостей, які визначають його як міцний інструмент для фронтенд розробки:

1. Компонентна архітектура: Angular використовує компонентний підхід, де кожен компонент має свій HTML шаблон і клас TypeScript для логіки. Це забезпечує високий рівень повторного використання коду і легшу підтримку.
2. Двосторонній дата байндинг: Angular автоматично синхронізує дані між моделлю і представленням, що дозволяє розробникам заощадити час і зусилля, не виконуючи додаткові маніпуляції для відображення оновлень у користувальницькому інтерфейсі.
3. Dependency Injection (DI): Angular використовує систему впровадження залежностей, яка спрощує створення і управління залежностями між різними класами і модулями.
4. Маршрутизація: Angular має вбудований механізм маршрутизації, який дозволяє керувати навігацією в односторінкових застосунках (SPA).
5. Модульність: Завдяки модульному підходу, Angular дозволяє розбивати застосунок на функціональні блоки, що спрощує тестування і підтримку.

Angular використовує TypeScript як основу, яка допомагає підвищити продуктивність розробки завдяки компіляції до JavaScript і підтримці

строгих типів, що забезпечує краще виявлення помилок на етапі розробки. Сам фреймворк включає набір інструментів та бібліотек, таких як:

- Angular CLI: інструмент командного рядка для створення проектів, додавання файлів і виконання різноманітних розробницьких задач.
- RxJS: бібліотека для асинхронного програмування за допомогою спостережуваних послідовностей, яка інтенсивно використовується в Angular для управління подіями та даними.
- Angular Material: набір готових до використання компонентів інтерфейсу згідно з принципами Material Design.

Angular є вибором багатьох великих компаній і корпорацій через його надійність і підтримку з боку Google. Серед відомих користувачів Angular можна відзначити:

– Google
– Microsoft
– IBM
– PayPal
– Samsung
– Forbes

Створення проекту в Angular розпочинається з встановлення інструменту командного рядка, званого Angular CLI. Це інструмент допомагає спростити багато задач, пов’язаних з розробкою веб-додатків. Ось основні кроки (див. рис. 2.7), які виконуються для розгортання нового проекту на Angular:

1. Встановлення Angular CLI: Завантаження та встановлення CLI забезпечує доступ до інструментів для створення та управління проектом.
2. Створення нового проекту: За допомогою команди ng new project-name створюється новий проект з усіма необхідними файлами.

3. Додавання компонентів: Використання команди ng generate component component-name дозволяє швидко створювати нові компоненти з відповідними файлами для HTML, CSS та TypeScript.
4. Розробка та налаштування: Після створення базової структури, можна приступати до налаштування та реалізації бізнес-логіки додатку.

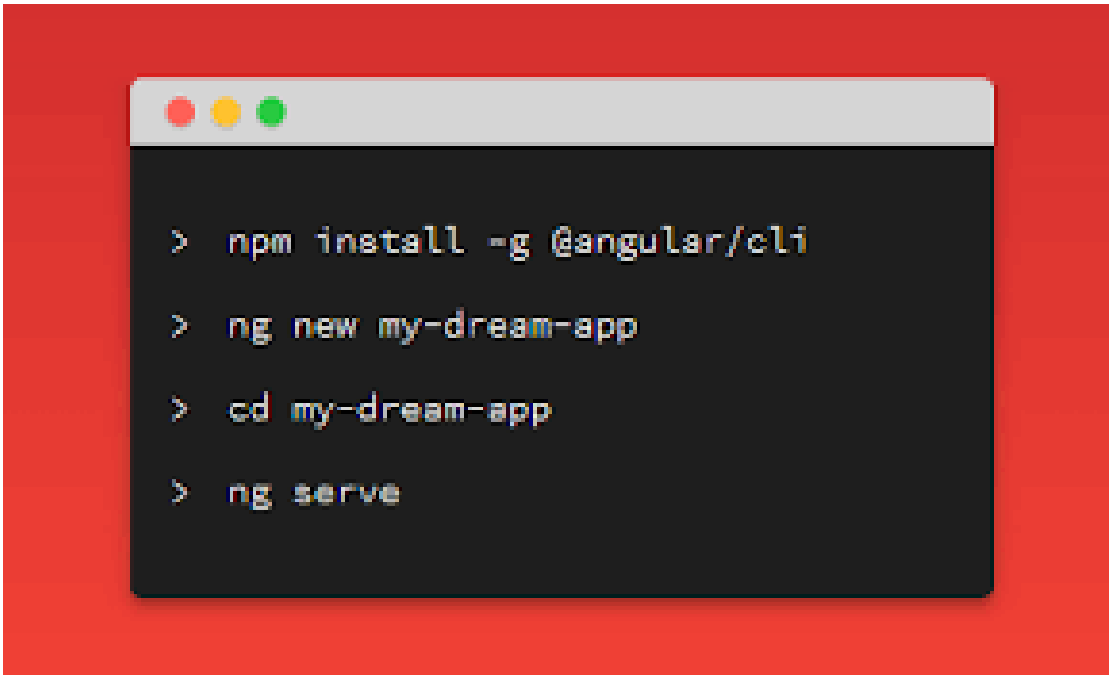


Рис. 2.7 Ілюстрація команд в терміналі для створення Angular проекту

Ці кроки формують фундамент роботи з Angular і допомагають ефективно організувати процес розробки.

2.4 Система управління базою даних

PostgreSQL — це потужна об'єктно-реляційна система управління базами даних (СУБД) (див. рис. 2.8), відома своєю надійністю, гнучкістю та підтримкою широкого спектру даних, включаючи структуровані, напівструктуровані та навіть неструктуровані типи. Розроблена як проєкт з відкритим вихідним кодом, PostgreSQL підтримується великою спільнотою розробників і експертів з баз даних, що забезпечує її постійне оновлення та оптимізацію. PostgreSQL часто

вибирають для великих і вимогливих додатків через її здатність ефективно масштабуватися від невеликих до величезних обсягів даних.

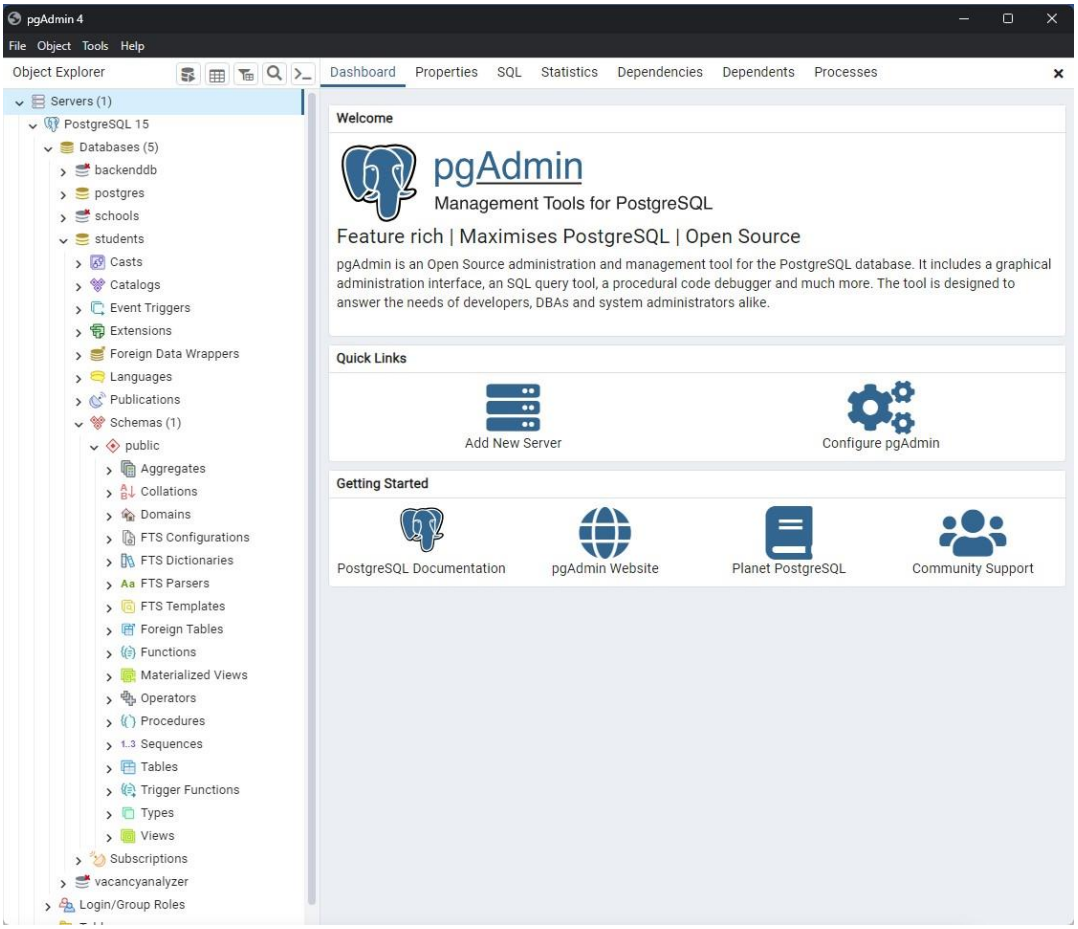


Рис. 2.8 Інтерфейс PostgreSQL для керування базами даних

Перш ніж глибше зануритись у технічні деталі, важливо зазначити, що PostgreSQL вирізняється серед інших СУБД завдяки кільком ключовим характеристикам. Ці особливості не тільки покращують продуктивність, але й надають високу гнучкість і масштабованість, які є критично важливими для сучасних додатків. Ось декілька з найбільш значущих характеристик:

- Підтримка складних запитів
- Розширені можливості індексації
- Підтримка транзакцій
- Многоверсіонність (MVCC)

– Інтеграція нових функцій

Окрім вже зазначених характеристик, PostgreSQL вирізняється також високим рівнем надійності та безпеки. Система забезпечує розширені можливості управління доступом та шифрування даних, що робить її ідеальним вибором для організацій, які мають суворі вимоги до захисту інформації.

2.5 Документування та система контролю версій

Документація та контроль версій відіграють критично важливу роль у процесі розробки програмного забезпечення. Для ефективного управління проектами та забезпечення високої якості кінцевого продукту, я використовував декілька спеціалізованих інструментів, кожен з яких вирішує свої задачі в проекті.

Trello — це інтуїтивно зрозумілий інструмент для управління проектами, який допомагає організувати проектні завдання у вигляді карток, що розміщуються на дошках. Кожна картка може містити деталі задачі, чек-листи, призначення для конкретних осіб, терміни виконання та мітки. Цей інструмент дозволив мені ефективно координувати процес розробки та забезпечити прозорість завдань для всієї команди.

GitHub — один з найпопулярніших сервісів контролю версій, що використовує систему Git. Це ключовий інструмент для співпраці в команді, що дозволяє кожному учаснику проекту працювати з кодом незалежно, вносити зміни та об'єднувати їх через пул-реквести, а також відслідковувати історію змін і відновлювати попередні версії при необхідності (див. рис. 2.9). Використання GitHub також полегшує процес ревізії коду та управління завданнями.

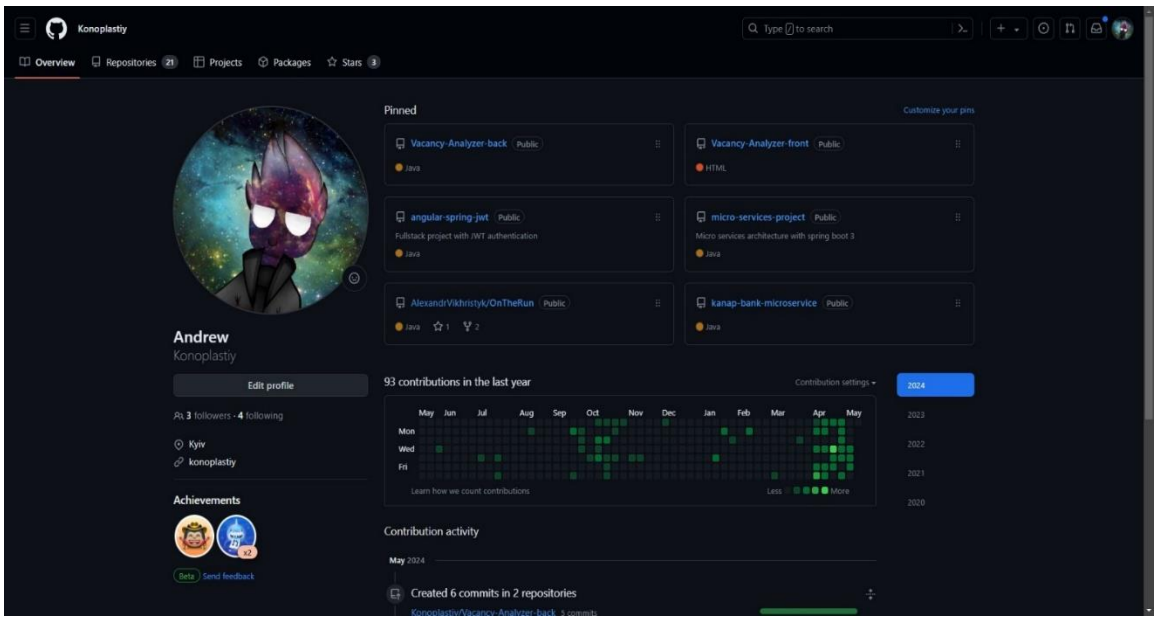


Рис. 2.9 Інтерфейс профілю на GitHub

Figma — є одним з передових інструментів для дизайну інтерфейсів. Це веб-базований додаток, що дозволяє дизайнерам та розробникам спільно працювати над дизайном макетів веб-застосунків у реальному часі. Figma забезпечує гнучкість у створенні, прототипуванні та передачі дизайну, що істотно спрощує процес передачі дизайну від дизайнера до розробника. Приклад можно подивитися на малюнку 3.10.

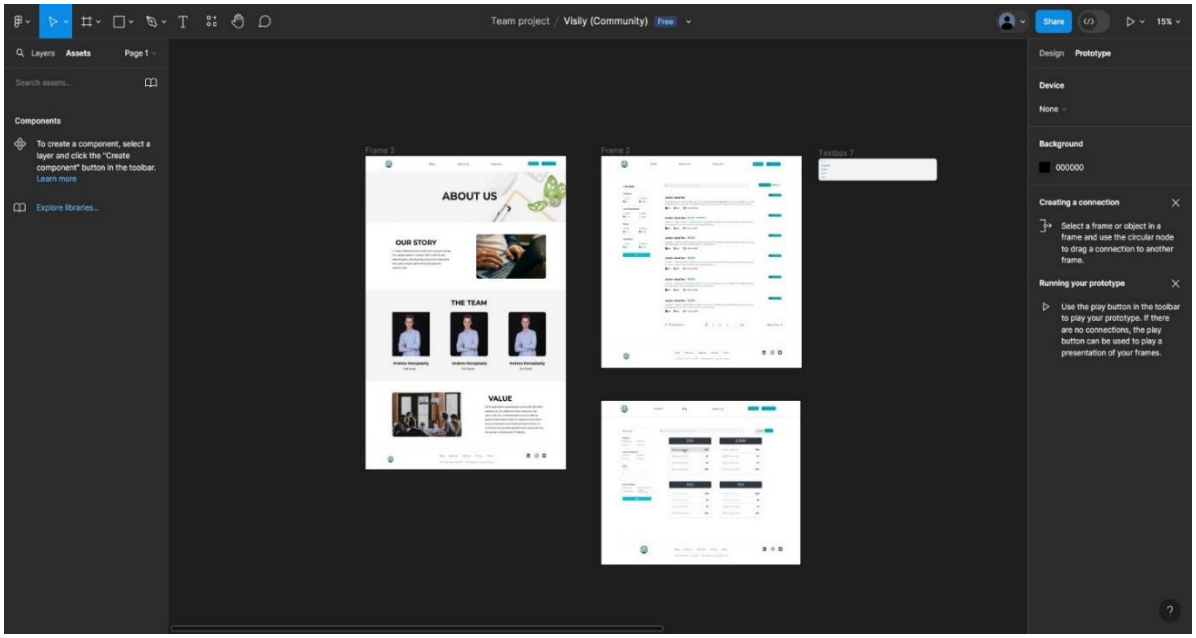


Рис. 2.10 Приклад розробки дизайну для веб-застосунку

Комбінація цих інструментів дозволяє ефективно управляти розробкою проектів, мінімізувати ризики, пов'язані з помилками в коді, та забезпечити високу якість і оперативність реалізації проектних рішень. Використання сучасних технологій та методик управління проектами допомагає тримати. весь процес під контролем і забезпечує успішну взаємодію всіх учасників проекту.

3 ОПИС РОЗРОБКИ ДОДАТКУ

3.1 Опис сторінок додатку та їх взаємодії

Мій веб-додаток складається з декількох ключових сторінок, кожна з яких відіграє свою унікальну роль в структурі сайту і задовольняє специфічні потреби користувачів. Сторінки організовані таким чином, щоб забезпечити легке навігування і інтуїтивно зрозуміле використання ресурсів сайту.

Список сторінок:

- Головна сторінка
- Блог
- Про нас
- Умови користування
- Конфіденційність

Головна сторінка - є центральним елементом сайту (див. рис. 3.1), де реалізована основна логіка: список вакансій з можливістю фільтрації та пошуку. Ця сторінка дозволяє користувачам швидко зорієнтуватися у вакансіях, які відповідають їхнім критеріям за допомогою інтуїтивно зрозумілих фільтрів (наприклад, за ключовими словами, регіоном, спеціалізацією). Це перше місце, куди потрапляють користувачі, тому важливо забезпечити позитивне перше враження і зручність використання.

Блог сторінка - сторінка зі списком блог-постів (див. рис. 3.2), де кожен пост має короткий опис. Користувачі можуть натиснути на заголовок посту, щоб перейти до повної версії статті та читати її детальніше.

Про нас сторінка - містить інформацію про компанію, її історію, місію та цілі (див. рис. 3.3). Ця сторінка допомагає користувачам зрозуміти, хто вони та чому варто довіряти компанії.

Умови користування сторінка - надає юридичну інформацію про правила та умови використання сайту (див. рис. 3.4), забезпечуючи користувачам знання про їхні права та обов'язки під час користування сайтом.

Конфіденційність сторінка - описує політику конфіденційності компанії (див. рис. 3.5), включаючи як і для чого збираються особисті дані користувачів, а також заходи, що вживаються для їх захисту.

Кожна сторінка виконує важливі функції і забезпечує користувачів необхідною інформацією та інструментами для ефективної взаємодії з сайтом.

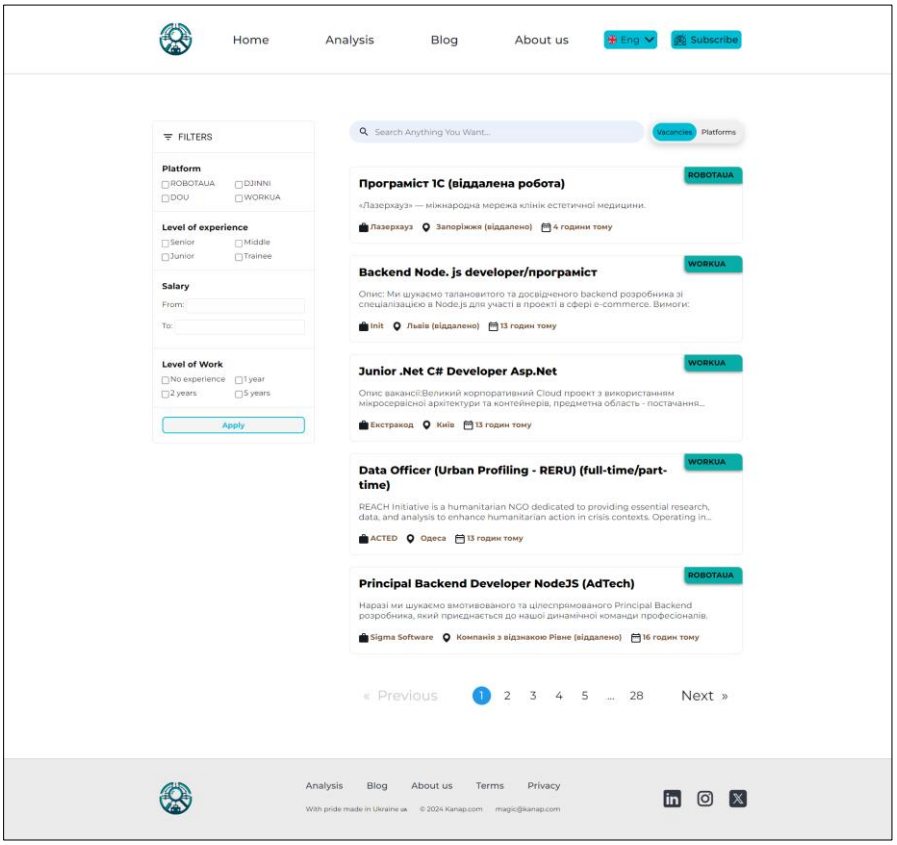


Рис. 3.1 Головна сторінка Vacancy-Analyzer

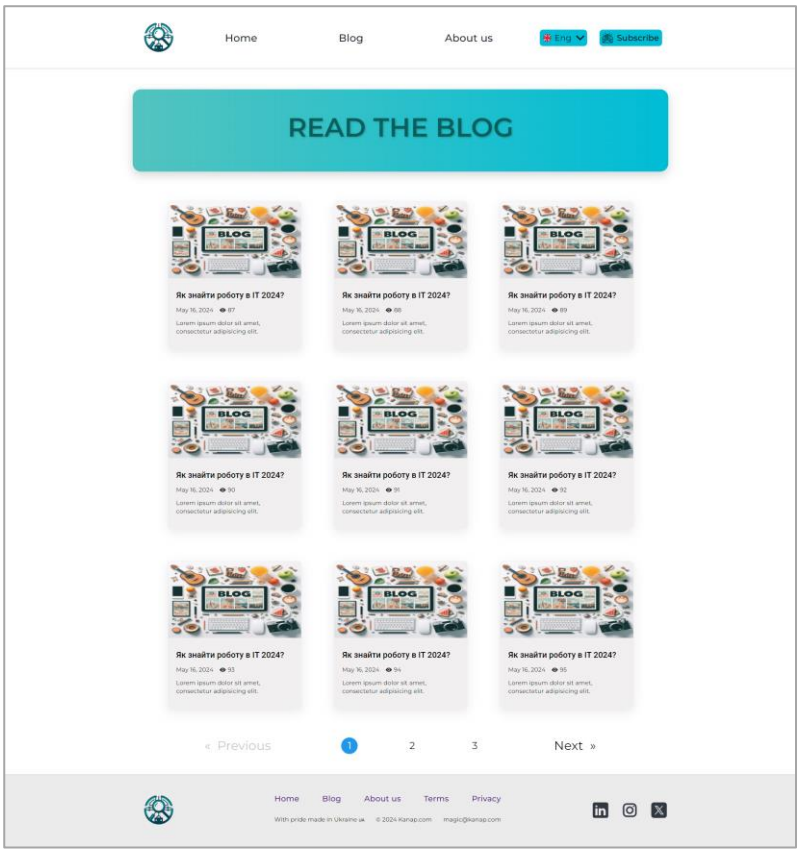


Рис. 3.2 Сторінка Блогу

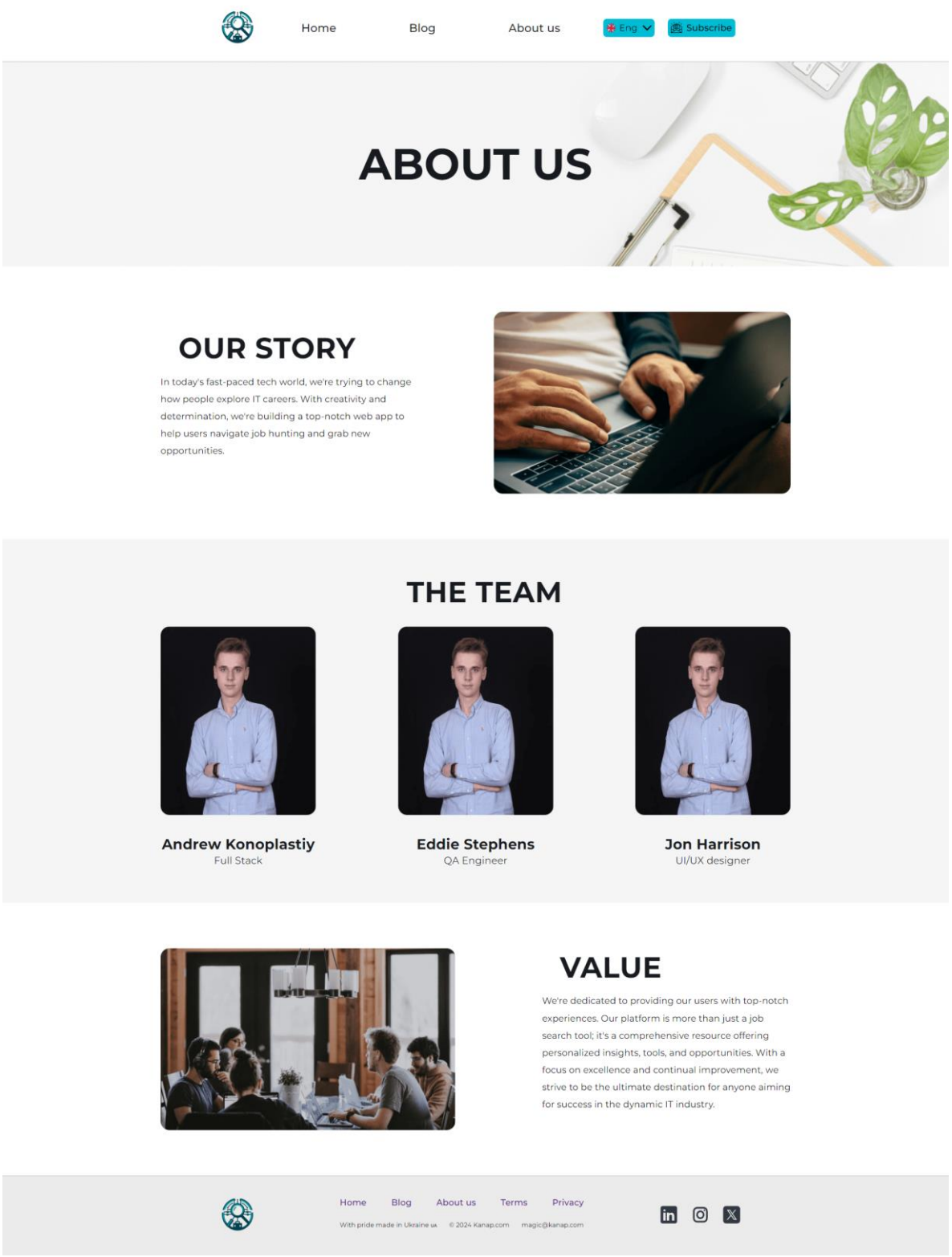


Рис. 3.3 Сторінка Про нас

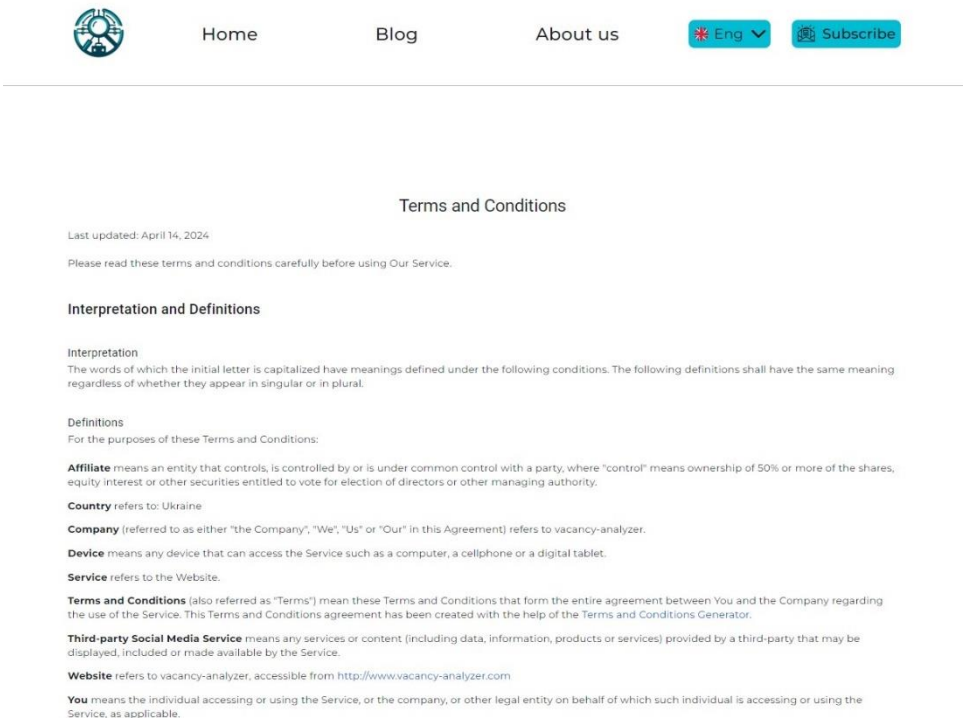


Рис. 3.4 Сторінка Умови користування

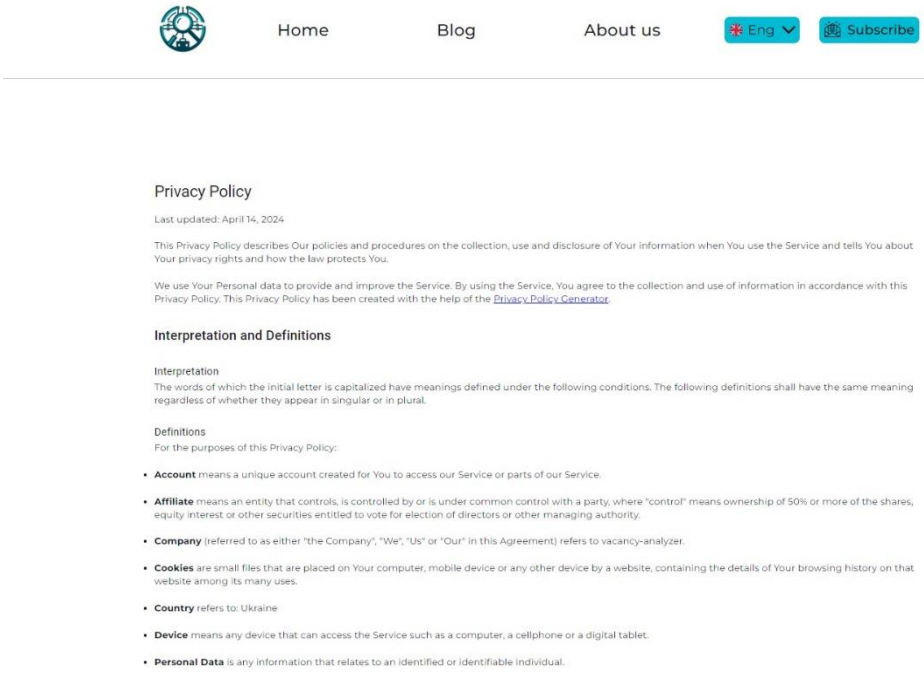


Рис. 3.5 Сторінка Конфіденційність

3.2 Архітектура

Архітектура програмного забезпечення грає вирішальну роль у забезпеченні основи для розвитку та скалабельності продукту. Вона визначає як компоненти програми взаємодіють між собою, а також з іншими системами, включаючи апаратні ресурси, зовнішні сервіси, і навіть кінцевих користувачів. Правильний вибір архітектури може значно поліпшити продуктивність, масштабованість та легкість управління продуктом.

Основні типи архітектури, які застосовуються у розробці програмного забезпечення, включають:

- Монолітична архітектура
- Мікросервісна архітектура
- Шарова архітектура (n-tier)

Монолітична архітектура — це традиційний стиль розробки програмного забезпечення, де всі компоненти програми об'єднані в єдиний неділимий блок або модуль (див. рис. 3.6). Цей підхід характеризується створенням одного ізольованого застосунка, де бізнес-логіка, база даних та інтерфейс користувача тісно інтегровані один з одним.

Всі компоненти програми створюються одночасно, що полегшує інтеграцію та процес тестування, адже відпадає необхідність синхронізувати окремі сервіси чи модулі. Розгортання моноліту відбувається просто, бо потрібно управляти тільки однією програмною сукупністю або артефактом. Це сприяє зручності розгортання в стандартних виробничих умовах. Однією з переваг є те, що всі необхідні залежності та бібліотеки вже включені до проекту, що мінімізує ризики пов'язані з конфліктами залежностей та полегшує процес управління версіями.

Ця архітектура має декілька значних недоліків:

- Масштабування: Монолітичні системи часто ускладнюють масштабування окремих компонентів, вимагаючи більше ресурсів, ніж необхідно.
- Оновлення та обслуговування: Зміни у системі можуть потребувати повного перезапуску, що збільшує ризик збоїв і простою.
- Негнучкість: Внесення нових технологій або оновлень у моноліт може бути складним і затратним, що гальмує інновації.

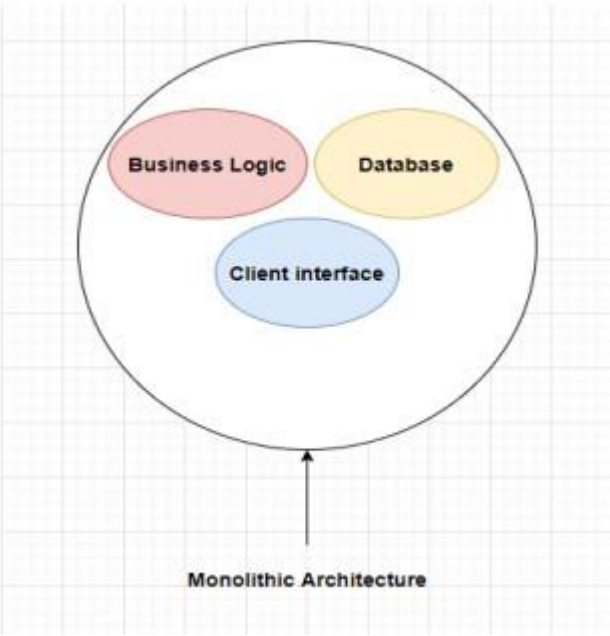


Рис. 3.6 Схема монолітної архітектури

Розглянемо веб-додаток для управління проектами. Всі основні аспекти програми, такі як управління завданнями, користувачами, ресурсами та звітами, інтегровані в одне рішення. Користувачі взаємодіють з системою через єдиний веб-інтерфейс, де можна створювати нові проекти, призначати завдання та відслідковувати прогрес.

Мікросервісна архітектура — це підхід до розробки програмного забезпечення, який полягає у створенні додатка як набору невеликих, автономних служб, кожна з яких виконує певні функції і спілкується з іншими через добре визначені API (див. рис. 3.7). Кожен мікросервіс працює незалежно і може бути розроблений, розгорнутий та масштабований окремо.

Мікросервіси підтримують розподіл бізнес-логіки на менші, взаємопов'язані частини, які можуть бути розгорнуті незалежно. Це дозволяє командам розробників працювати паралельно, покращує модульність і спрощує інтеграцію та автоматизоване тестування. Кожен мікросервіс часто має свою власну базу даних, що забезпечує локалізацію даних та меншу залежність сервісів один від одного.

Переваги:

- Гнучкість у розгортанні: Мікросервіси можуть бути розгорнуті незалежно, що знижує ризик для всієї системи при оновленні окремих сервісів.
- Легкість масштабування: Кожен мікросервіс можна масштабувати незалежно, що дозволяє ефективніше використовувати ресурси і краще відповідати на зміни в навантаженні.
- Відмовостійкість: Оскільки кожен мікросервіс є незалежним, збій одного сервісу менш імовірно призведе до збою всієї системи.
- Технологічна гнучкість: Різні мікросервіси можуть використовувати різні технології та бази даних, що надає можливість вибирати найкращі інструменти для конкретних завдань.

Недоліки:

- Складність управління: Управління великою кількістю мікросервісів може бути складним, особливо з точки зору моніторингу, логування та забезпечення безпеки.
- Проблеми з продуктивністю: Залежно від того, як виконується взаємодія між сервісами, мережева затримка і загальні витрати на комунікацію можуть вплинути на продуктивність.
- Складність транзакцій: Управління транзакціями, які розподілені між декількома мікросервісами.

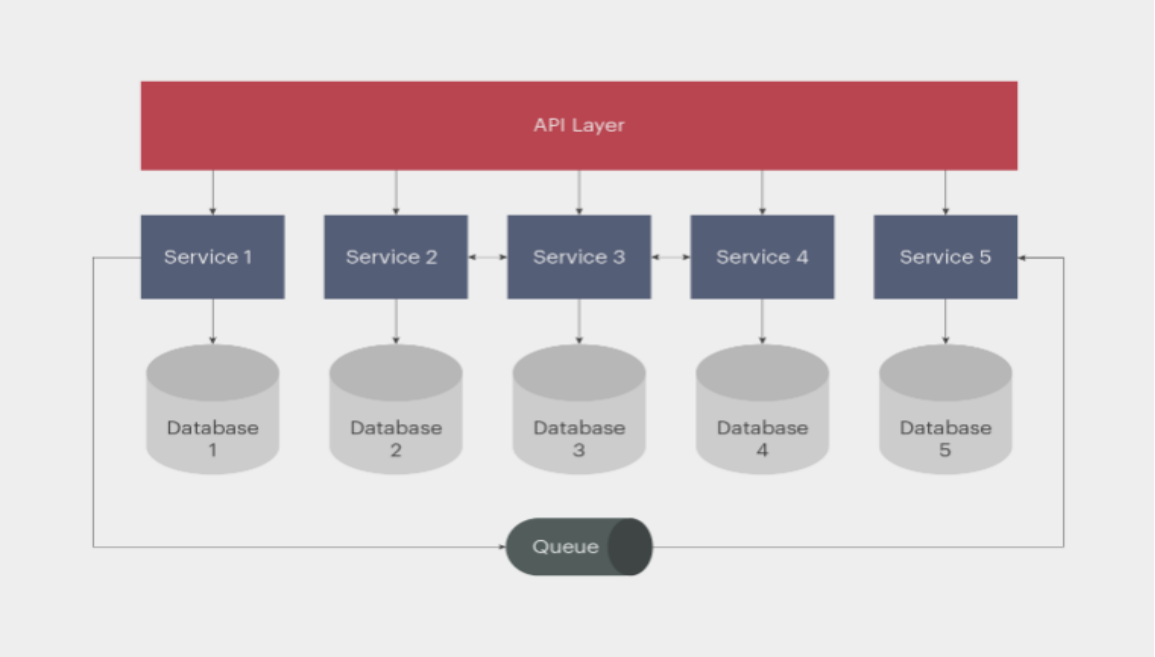


Рис. 3.6 Схема мікросервісної архітектури

Розглянемо веб-додаток для управління проектами, реалізований за принципами мікросервісної архітектури. В цьому додатку, кожен аспект управління проектом — від створення завдань до ведення звітності — виконується незалежними сервісами, кожен з яких фокусується на конкретній доменній області.

Користувачі взаємодіють із системою через веб-інтерфейс, який забезпечує єдину точку доступу до всіх сервісів. Коли користувач створює новий проект, запит відправляється до мікросервісу управління проектами, який займається реєстрацією проекту та його атрибутів. Коли ж користувач додає завдання до проекту, ці дані обробляються окремим мікросервісом завдань, який спеціалізується на управлінні завданнями, їх статусами та пріоритетами.

Шарова архітектура, відома також як n-tier архітектура, — це підхід до розробки програмного забезпечення, де додаток розділений на логічно розгороджені шари або рівні (див. рис. 3.8). Кожен рівень відповідає за певну функціональність, що забезпечує модульність і легкість управління. Мікросервіси підтримують розподіл бізнес-логіки на менші, взаємопов'язані частини, які можуть бути розгорнуті незалежно. Це дозволяє командам розробників працювати паралельно.

В шаровій архітектурі компоненти додатка розділені на шари, такі як презентаційний шар, бізнес-логіка, і база даних. Це розділення дозволяє зосередитись на розвитку та оптимізації окремих аспектів системи незалежно від інших.

Переваги:

- Відокремлення відповідальності: Кожен шар зосереджений на певній функціональності, що полегшує управління та технічне обслуговування.
- Гнучкість у виборі технологій: Можливість використання різних технологій в різних шарах без впливу на інші частини додатка.
- Легкість розширення та масштабування: Шари можуть бути масштабовані незалежно від інших, що забезпечує ефективне використання ресурсів.

Недоліки:

- Складність архітектури: Шарова структура може зробити архітектуру складнішою і трудомісткою в підтримці.
- Продуктивність: Передача даних між шарами може створювати затримки, особливо при неналежному проектуванні.
- Жорсткість: Іноді важко реалізувати зміни, які вимагають взаємодії декількох шарів, особливо в старих і великих системах.

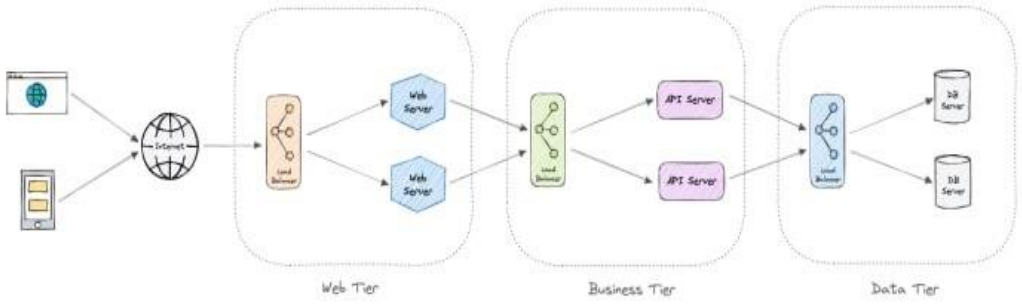


Рис. 3.7 Схема шарової архітектура

Розглянемо веб-додаток для управління проектами, створений на основі шарової архітектури, де користувачі взаємодіють із системою через

презентаційний шар, що забезпечує веб-інтерфейс для створення та управління проектами. Запити від користувачів обробляються бізнес-логічним шаром, який виконує всі необхідні бізнес-правила, такі як перевірка дозволів та управління ресурсами, а шар даних відповідає за взаємодію з базою даних для збереження та витягування інформації. Це розділення ролей між шарами дозволяє ефективно масштабувати кожен компонент окремо та спрощує процес розробки та підтримки системи.

3.3 Вимоги до програмного забезпечення

Вимоги до програмного забезпечення визначають ключові функції та технічні специфікації, які повинен підтримувати веб-сайт для пошуку та фільтрації вакансій. Цей документ слугує основою для розробників і проектувальників, забезпечуючи чіткі орієнтири для створення ефективного та користувацьки дружнього інтерфейсу. За допомогою деталізації кожного аспекту вимог, можливо забезпечити високу якість розробки та впевненість у відповідності кінцевого продукту до очікувань користувачів. Ці вимоги можна візуально представити у вигляді діаграми прецедентів (див. рис. 3.9).

Рис. 3.9 Діаграма прецедентів

Функціональність пошуку вакансій:

– Реалізація пошуку за ключовими словами з використанням передових алгоритмів пошуку для високої точності результатів.

Фільтр для сортування вакансій:

– Надання користувачам опцій для сортування результатів пошуку за різними параметрами, зокрема рівнем досвіду, зарплатою та іншими параметрами.

Зміна мови:

– Підтримка мінімум двох мов для забезпечення доступності сайту для більшої аудиторії.

– Автоматичне визначення мови за налаштуваннями браузера користувача з можливістю ручного вибору в налаштуваннях профілю.

Підписка на новини:

- Інтеграція форми підписки у користувацький інтерфейс для зручності користувачів.
- Розробка системи автоматичного надсилання електронних листів, яка інформуватиме користувачів про останні оновлення та вакансії, які відповідають їхнім критеріям пошуку.
- Впровадження заходів безпеки для захисту персональних даних користувачів та запобігання розсилці спаму.

Інтерфейс користувача:

- Розробка чистого, інтуїтивно зрозумілого дизайну, який адаптується до різних пристроїв та розмірів екранів.
- Забезпечення високої швидкості завантаження сторінок завдяки оптимізації зображень та скриптів.

Вимоги до тестування:

- Реалізація комплексного плану тестування, що включає юніт-тести, інтеграційні тести, системні тести та приймальні тести для забезпечення якості на всіх рівнях розробки.

Дотримання встановлених вимог до програмного забезпечення є ключовим для успішного запуску веб-сайту, який би задовольняв потреби сучасного ринку праці та користувачів, що шукають роботу. Впровадження цих вимог вимагатиме тісної співпраці між розробниками, дизайнерами та аналітиками для створення продукту, який не тільки відповідає технічним стандартам, але й пропонує користувачам неперевершений досвід використання.

3.4 Проєктування бази даних

Розробка структури бази даних відіграє вирішальну роль у створенні веб-додатку, оскільки добре спланована база даних сприяє надійній взаємодії користувачів з системою. Описана нижче діаграма (див. рис. 3.10) ілюструє структуру та зв'язки між таблицями. Ці відомості допоможуть краще зрозуміти, як дані обробляються та управляються всередині платформи.

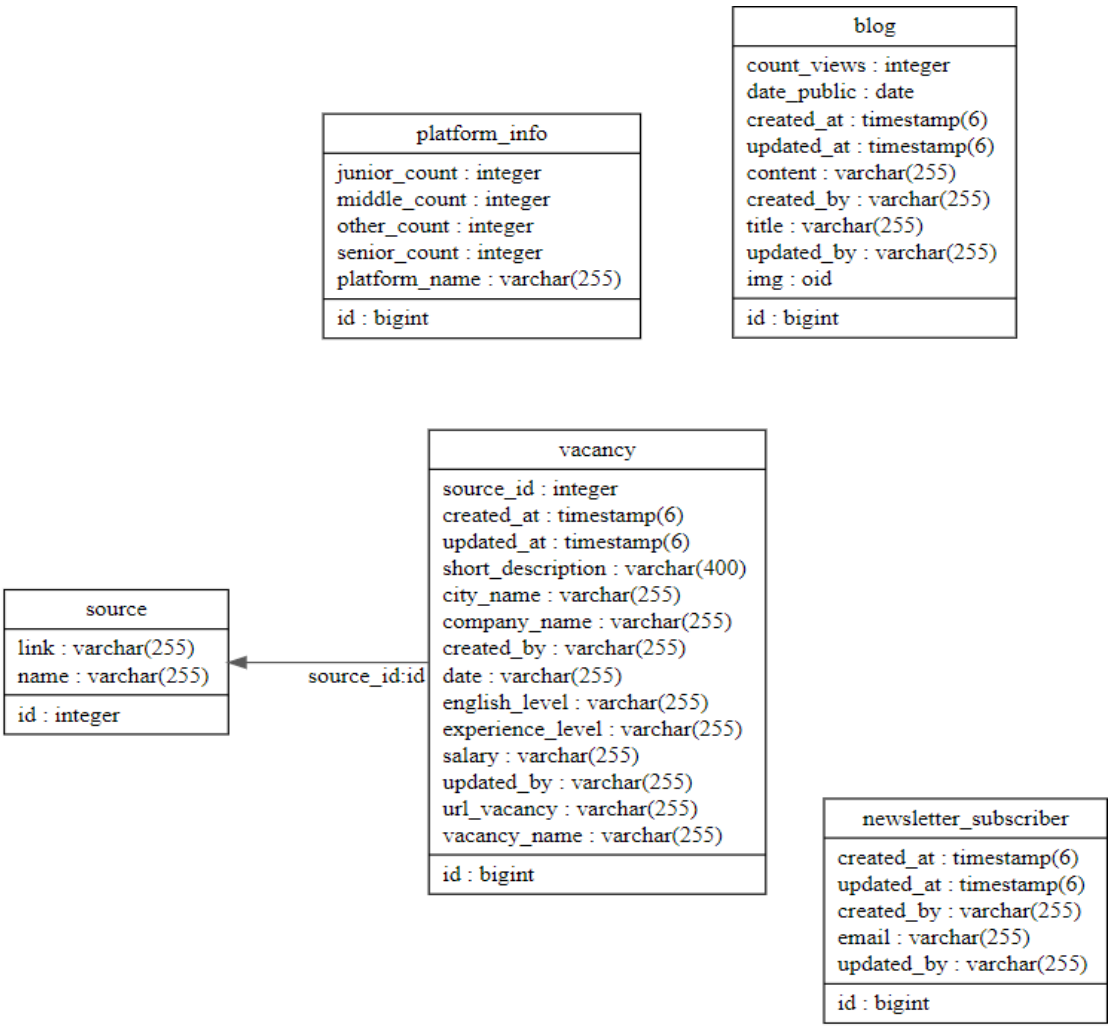


Рис. 3.10 Схема бази даних для веб-застосунку

Таблиця blog використовується для управління контентом блогів на веб-платформі. Вона зберігає інформацію про кожен блог-пост, включаючи:

- count_views (integer): Кількість переглядів поста.
- date_public (date): Дата публікації поста.
- created_at (timestamp(6)): Відмітка часу створення поста.
- updated_at (timestamp(6)): Відмітка часу оновлення поста.
- content (varchar(255)): Зміст поста.
- title (varchar(255)): Заголовок поста.
- created_by (varchar(255)): Користувач, який створив пост.
- updated_by (varchar(255)): Користувач, який останній оновив пост.
- img (oid): Посилання на зображення, асоційоване з постом.
- id (bigint): Унікальний ідентифікатор поста.

Таблиця newsletter_subscriber використовується для зберігання інформації про підписників на новини. Основні поля таблиці включають:

- email (varchar(255)): Електронна адреса підписника.
- created_at (timestamp(6)): Відмітка часу створення запису.
- updated_at (timestamp(6)): Відмітка часу оновлення запису.
- created_by (varchar(255)): Користувач, який створив запис.
- updated_by (varchar(255)): Користувач, який останній оновив запис.
- id (bigint): Унікальний ідентифікатор підписника.

Таблиця platform_info зберігає статистичні дані про користувачів платформи, що дозволяє аналізувати демографічну структуру аудиторії. Вона включає такі поля:

- junior_count (integer): Кількість junior користувачів.
- middle_count (integer): Кількість middle користувачів.
- senior_count (integer): Кількість senior користувачів.
- other_count (integer): Кількість користувачів інших категорій.
- platform_name (varchar(255)): Назва платформи.
- id (bigint): Унікальний ідентифікатор інформації про платформу.

Таблиця vacancy призначена для зберігання інформації про доступні вакансії. Основні поля таблиці:

- source_id (integer): Ідентифікатор джерела вакансії.
- created_at (timestamp(6)): Відмітка часу створення вакансії.
- updated_at (timestamp(6)): Відмітка часу оновлення вакансії.
- short_description (varchar(400)): Короткий опис вакансії.
- city_name (varchar(255)): Місто розташування вакансії.
- company_name (varchar(255)): Назва компанії, що пропонує вакансію.
- created_by (varchar(255)): Користувач, який створив вакансію.
- date (varchar(255)): Дата публікації вакансії.
- english_level (varchar(255)): Рівень володіння англійською мовою.

- experience_level (varchar(255)): Рівень досвіду, необхідний для вакансії.
- salary (varchar(255)): Зарплата.
- updated_by (varchar(255)): Користувач, який останній оновив вакансію.
- url_vacancy (varchar(255)): URL вакансії.
- vacancy_name (varchar(255)): Назва вакансії.
- id (bigint): Унікальний ідентифікатор вакансії.

Таблиця source використовується для зберігання інформації про джерела вакансій:

- link (varchar(255)): Посилання на джерело.
- name (varchar(255)): Назва джерела.
- id (integer): Унікальний ідентифікатор джерела.

Ці таблиці разом створюють архітектуру бази даних, яка підтримує ключові операції веб-додатку та забезпечує оперативний доступ до важливих даних для його ефективної функціональності.

3.5 Розробка основного функціоналу проекту

Процес парсингу вакансій є ключовою складовою системи, оскільки він дозволяє збирати актуальні дані про робочі місця з різноманітних джерел. На рисунку 3.11 зображена структура основної логіки Back end частини. Кожен сервіс у системі, використовує многопоточний підхід для оптимізації збору даних, що значно покращує швидкість і ефективність обробки інформації. Основна мета цих сервісів полягає в автоматизації процесів збору вакансій, їх аналізу та класифікації з подальшим збереженням у базі даних.

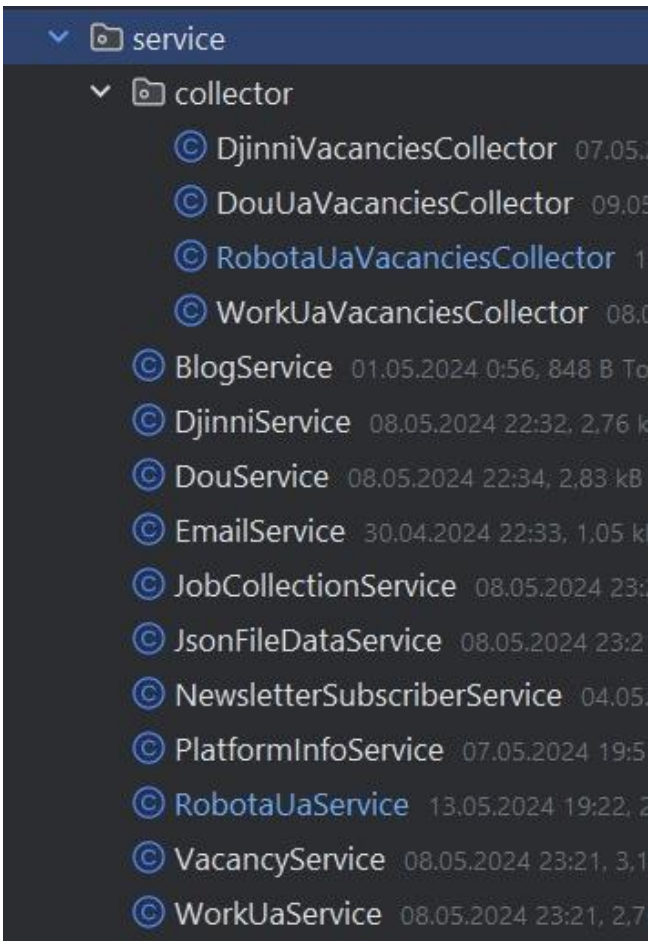


Рис. 3.11 Структура основної логіки

Процес роботи парсингових сервісів розпочинається з ініціації сервісу, як на прикладі RobotaUaService, що включає визначення джерела даних і отримання необхідних URL для збору вакансій. Цей процес запускається асинхронно для забезпечення високої продуктивності системи. Далі, використовуючи ExecutorService, створюється пул потоків, де кожен потік має завдання парсингу, представлені екземплярами класу RobotaUaVacanciesCollector, які реалізують інтерфейс Callable. Це дозволяє збирати дані паралельно та ефективно.

В процесі збору даних, колектори виконують HTTP-запити до вказаних URL, обробляючи веб-сторінки за допомогою бібліотеки Jsoup для витягування інформації про вакансії. Звідти, дані такі як назва вакансії, компанія, місце розташування, зарплата та інші деталі, екстрагуються та перетворюються в структурований формат. Всі зібрані вакансії потім конвертуються в об'єкти DTO (Data Transfer Object), які мапляться в ентиті бази даних за допомогою

VacancyMapper. Заключний етап включає збереження остаточно зібраних даних у базі даних через VacancyRepository.

Управління помилками відіграє критичну роль у забезпеченні стабільності сервісу. Система активно логує всі помилки і виняткові ситуації, що виникають під час виконання завдань. Це включає логування попереджень при помилках виконання та перериваннях потоків, що дозволяє оперативно реагувати на можливі проблеми та оптимізувати процеси.

Для кращого розуміння взаємодії між компонентами проекту, нижче наведено діаграму класів (див. рис. 3.12), яка відображає основні елементи системи і зв’язки між ними.

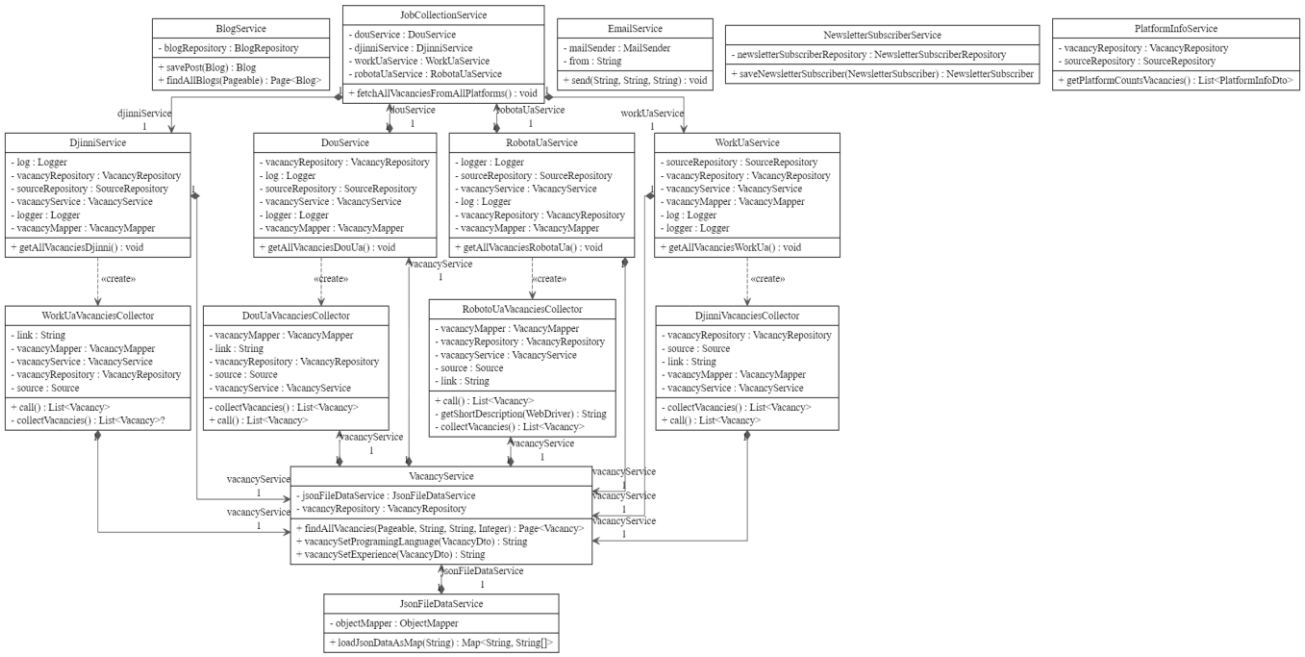


Рис. 3.12 Діаграма класів

Таким чином, система ефективно виконує свої функції на бекенді, забезпечуючи швидку обробку та надійну інтеграцію даних.

3.6 Тестування веб-додатку

При розробці програмного забезпечення, ефективне тестування є ключовим для забезпечення його надійності та якості. У цьому контексті, фреймворки Mockito і JUnit відіграли вирішальну роль. Mockito, зокрема, забезпечив можливість створення макетів залежностей. Це дозволило ізолювати тестування окремих компонентів, створюючи контрольоване середовище, де можна точно відтворити різні сценарії. Це забезпечило можливість випробувати поведінку системи під різними умовами без потреби взаємодії з реальними зовнішніми сервісами або базами даних.

Застосування JUnit сприяло структуруванню та автоматизації процесу тестування. З його допомогою можна було систематично перевіряти кожен аспект програмного забезпечення, від простих функцій до складних інтеграційних процесів. Обидва фреймворки разом забезпечили потужні інструменти для валідації коду перед його впровадженням у виробництво.

Завдяки інтеграції Mockito і JUnit у розробку, було досягнуто значних успіхів у забезпеченні якості програмного продукту. Всі модульні тести були успішно пройдені, що свідчить про високу стабільність та надійність розробленого рішення. Загальне покриття коду юніт-тестами склало близько 80%, що є свідченням ретельної перевірки більшості логічних компонентів програми. Це покриття дозволяє з великою впевненістю стверджувати про відсутність неочікуваних помилок та проблем у більшості сценаріїв використання програми.

Інтегроване тестування є ключовим не тільки для забезпечення високої якості програми, але й для побудови довіри серед користувачів і клієнтів. Воно гарантує надійність програмної продукції та мінімізує потенційні ризики, що можуть виникнути під час її розгортання у виробничому середовищі.

ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено веб-додаток для ефективного пошуку вакансій у галузі ІТ. Актуальність проекту визначається швидким розвитком ІТ-сектору і потребою фахівців знаходити актуальні пропозиції з працевлаштування, що відповідають їхнім професійним навичкам і інтересам.

Було проведено дослідження існуючих аналогів та інструментів пошуку вакансій, які виявили ряд недоліків та переваг. Це дозволило встановити ключові вимоги до нового веб-додатку, спрямовані на вдосконалення пошукового процесу та користувацького досвіду.

Розроблено детальні функціональні та нефункціональні вимоги до додатку. Значну увагу було приділено системі фільтрації вакансій та забезпеченню адаптивності інтерфейсу для різних пристроїв.

Для реалізації проекту були вибрані сучасні технології, зокрема мова програмування Java, фреймворк Spring для бекенду, а також Angular для фронтенду. Вибір заснований на швидкості, безпеку та масштабованості цих технологій.

Додаток було успішно розроблено та протестовано. Функціональне тестування показало, що всі основні вимоги були задоволені, і програма забезпечує стабільну роботу. Юніт-тести покрили 80% проекту, що підтвердило високу надійність розробки.

Додаток готовий до впровадження та використання кінцевими користувачами. Планується його даліше вдосконалення, включаючи розширення функціоналу блогу, оптимізацію алгоритмів пошуку і поліпшення механізмів зворотного зв'язку з користувачами.

У подальшому розвитку додатку планується інтеграція штучного інтелекту для прогнозування популярності технологій в ІТ-галузі, що забезпечить користувачам актуальну інформацію про майбутні тенденції і попит на певні навички. Використання алгоритмів машинного навчання дозволить аналізувати

великі масиви даних з оголошень про роботу та професійних публікацій, щоб точно ідентифікувати тренди і передбачити майбутні зміни у технологічних перевагах. Ця інформація стане основою для розробки рекомендаційних систем, які автоматично радитимуть користувачам відповідні курси навчання та сертифікації, підвищуючи їхні шанси на успішне працевлаштування.

Апробація результатів дослідження – робота успішно пройшла апробацію на Всеукраїнській науково-технічній конференції «Застосування програмного забзпечення в ІКТ» за темами: « Розробка програмного забезпечення для аналізу вакансій з використанням фреймворку Spring Boot та Angular», «Розробка веб-застосунку для прогнозування технологічних тенденцій у галузі інформаційних технологій з використанням штучного інтелекту, Spring boot та Angular»

ПЕРЕЛІК ПОСИЛАНЬ

1. Abhishek, Soumili Chandra, Soumili Chandra. Analysis and comparison of the spring framework, struts framework, vaadin framework, and play framework performance, used to create web applications in java. International journal of advanced research in science, communication and technology. 2023. P. 277–282. URL: https://doi.org/10.48175/ijarsct-9574 (date of access: 01.04.2024).
2. Progressive web apps development and analysis with angular framework and service worker for e-commerce system / Z. Tahir et al. 2021 IEEE international conference on computing (ICOCO), Kuala Lumpur, Malaysia, 17–19 November 2021. 2021. URL: https://doi.org/10.1109/icoco53166.2021.9673557 (date of access: 01.04.2024).
3. Padalkar S., Jaybhaye M. D., Jaybhaye S. M. Angular rxjs for e-commerce development. Intelligent manufacturing and energy sustainability. Singapore, 2023. P. 59–68. URL: https://doi.org/10.1007/978-981-99-6774-2_6 (date of access: 01.04.2024).
4. Algorithms and Data Structures / ред.: A. Lubiw, M. Salavatipour. Cham : Springer International Publishing, 2021. URL: https://doi.org/10.1007/978-3-030-83508-8 (дата звернення: 07.05.2024).
5. Progressive web apps development and analysis with angular framework and service worker for e-commerce system / Z. Tahir et al. 2021 IEEE international conference on computing (ICOCO), Kuala Lumpur, Malaysia, 17–19 November 2021. 2021. URL: https://doi.org/10.1109/icoco53166.2021.9673557 (дата звернення: 01.04.2024).
6. Garcia V. H. Introduction to Angular Framework. Getting Started with Angular. Berkeley, CA, 2023. С. 1–11. URL: https://doi.org/10.1007/978-1-4842-9206-8_1 (дата звернення: 06.05.2024).

7. Angular matrix framework for light trapping analysis of solar cells / Y. Li та ін. Optics Express. 2015. Т. 23, № 24. С. A1707. URL: https://doi.org/10.1364/oe.23.0a1707 (дата звернення: 19.05.2024).
8. Downey T. Spring Framework. Texts in Computer Science. Cham, 2021. С. 121–170. URL: https://doi.org/10.1007/978-3-030-62274-9_4 (дата звернення: 18.05.2024).
9. Hagos T. Beginning IntelliJ IDEA. Berkeley, CA : Apress, 2022. URL: https://doi.org/10.1007/978-1-4842-7446-0 (дата звернення: 18.05.2024).
10. Callaghan M. D. RxJS: To Use or Not to Use?. Angular for Business. Berkeley, CA, 2023. С. 137–146. URL: https://doi.org/10.1007/978-1-4842-9609-7_11 (дата звернення: 18.05.2024).
11. Publishing T. TypeScript Programming Language. Independently Published, 2019. 248 с.
12. Sacco A. Beginning Spring Data: Data Access and Persistence for Spring Framework 6 and Boot 3. Apress L. P., 2023.
13. Purohit R., Jain S., Gupta S. Role of Angular Framework in Web Development. International Journal of Advanced Research in Science, Communication and Technology. 2023. С. 114–127. URL: https://doi.org/10.48175/ijarsct-10731 (дата звернення: 18.05.2024).
14. Uzayr S. b. GitHub. Mastering Git. Boca Raton, 2022. С. 129–152. URL: https://doi.org/10.1201/9781003229100-6 (дата звернення: 18.05.2024).

ДОДАТОК А ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«Розробка програмного забезпечення для аналізу вакансій з використанням фреймворку Spring Boot, мовою Java»

Виконав студент 4 курсу
групи ПД-42
Коноплястий Андрій Русланович
Керівник роботи
Д.т.н, проф., завідувач кафедри ТЦР Жебка Вікторія Вікторівна

Київ – 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – спрощення процесу аналізу вакансій за допомогою веб-застосунку, який використовує мову Java та фреймворки Spring Boot і Angular.
- **Об'єкт дослідження** – аналіз вакансій та ринку праці в ІТ.
- **Предмет дослідження** – програмне забезпечення для аналізу, пошуку, та візуалізації вакансій, включаючи алгоритми обробки даних та інтерфейси користувача.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Дослідження ринку вакансій і визначення ключових викликів аналізу.
2. Провести огляд та порівняльний аналіз існуючих платформ для аналізу вакансій, таких як Djinni та Dou.ua, з метою ідентифікації можливих поліпшень для розроблюваного застосунку.
3. Сформулювати технічні вимоги та визначити основні функції застосунку, включаючи фільтрацію вакансій, пошук за текстом та підписку на новини.
4. Розробити базу даних і архітектуру компонентів системи для забезпечення взаємодії між модулями з використанням Spring Boot і Angular.
5. Здійснити кодування та інтеграцію системних компонентів, реалізувавши проектні рішення для підтримки функціоналу веб-застосунку.
6. Виконати модульне та інтеграційне тестування для виявлення та усунення помилок, забезпечуючи плавну взаємодію компонентів.
7. Провести аналіз результатів тестування для оцінки відповідності продукту встановленим цілям та визначити необхідні корективи.

3

АНАЛІЗ АНАЛОГІВ

Параметр	Djinni	Dou.ua	VA
Відображення вакансій з інших платформ	-	-	+
Розширений пошук за категоріями та фільтрами	+	+	+
Розділ блогів і статей	-	+	+
Статистика зарплат	+	+	+
Широкий спектр вакансій у різних галузях	-	-	+
Прямий контакт з роботодавцями	+	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Фукціональні вимоги:

1. Можливість змінювати мову інтерфейсу.
2. Пошук вакансії за допомогою фільтрів.
3. Текстовий пошук вакансій.
4. Можливість підписуватися на новини про вакансії.
5. Регулярне оновлення блогів з корисною інформацією про кар'єру та ринок праці.

Нефукціональні вимоги:

1. Інтерфейс користувача відрізняється легкістю використання та адаптивне верстання.
2. Завантаження сторінок застосунку менш ніж за 2 секунди.
3. Система підписки на новини має забезпечувати надійне та своєчасне розсилання інформації, а також можливість легкої відписки від розсилок.
4. Веб-застосунок має підтримувати англійську та українську мови, забезпечуючи точний переклад інтерфейсу та контенту для різних користувачів.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

spring

ANGULAR

PostgreSQL

Maven

HTML

Sass

TypeScript

mockito

POSTMAN

docker

GitHub

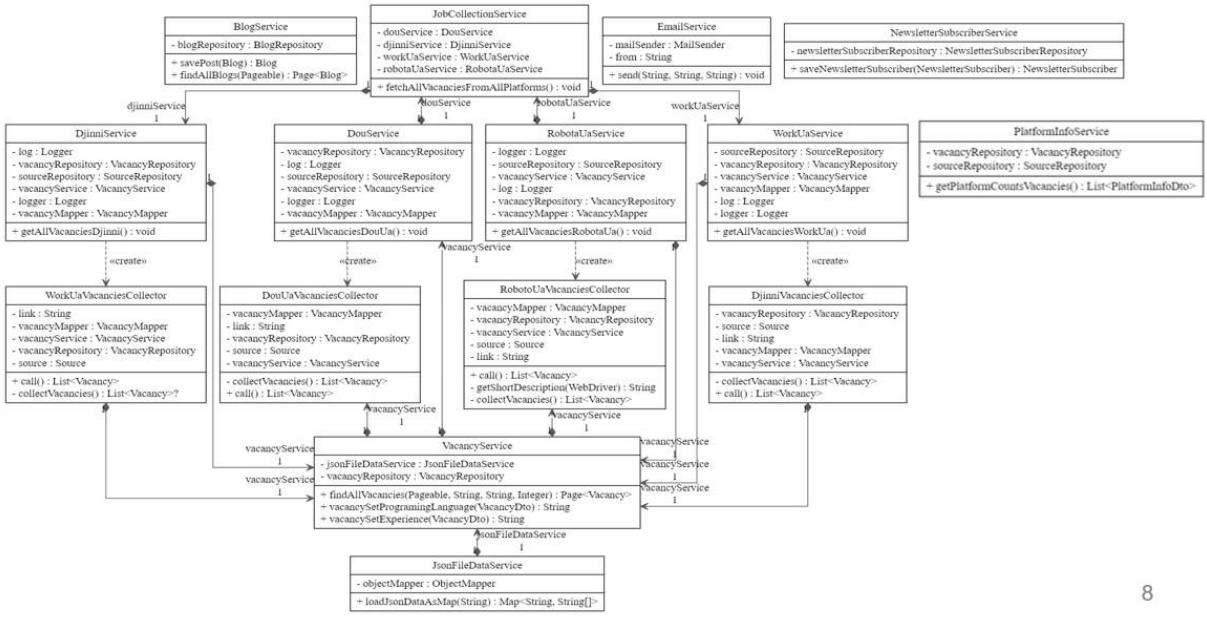
IntelliJ IDEA

6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

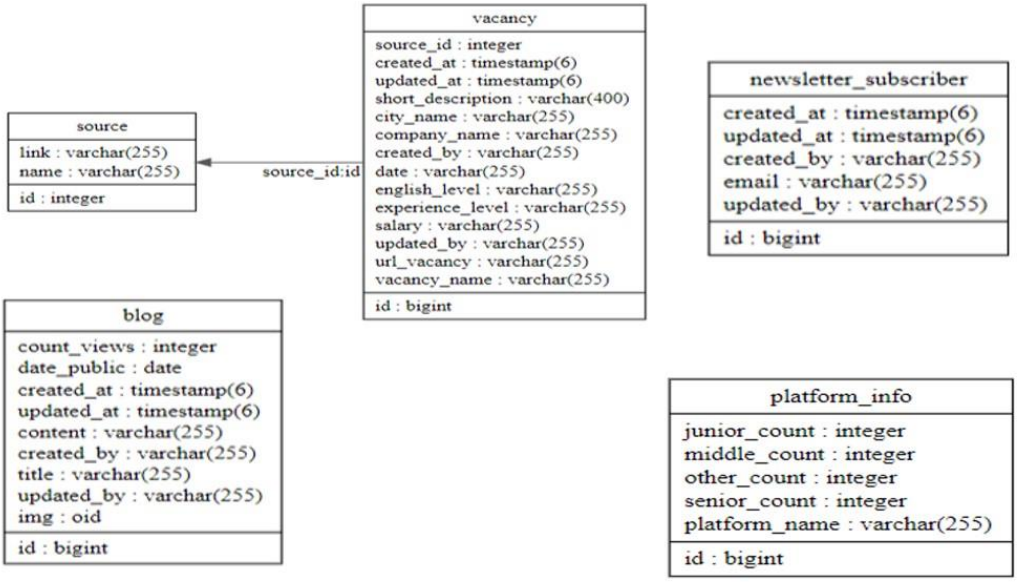
7

ДІАГРАМА КЛАСІВ



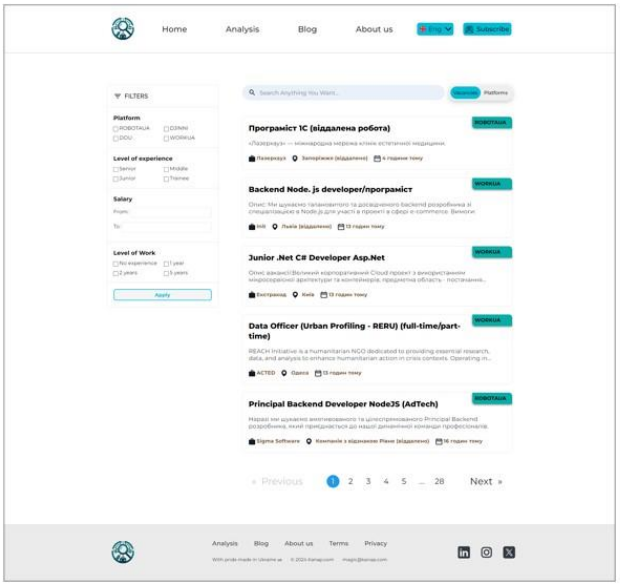
8

СТРУКТУРА БАЗИ ДАНИХ

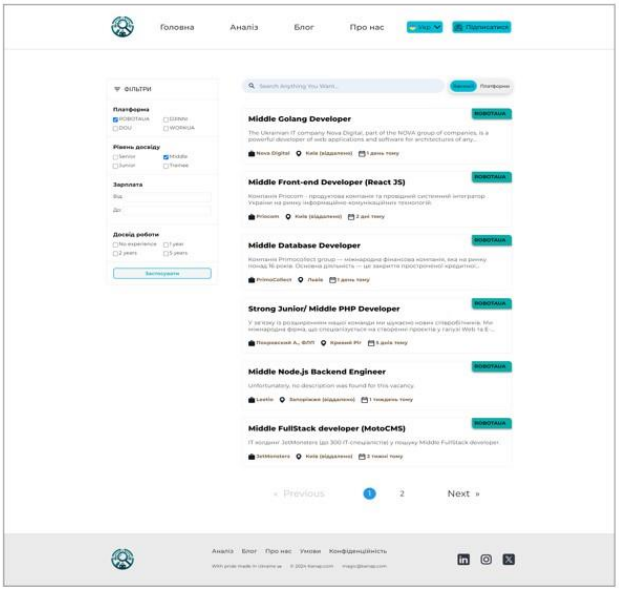


9

ЕКРАННІ ФОРМИ



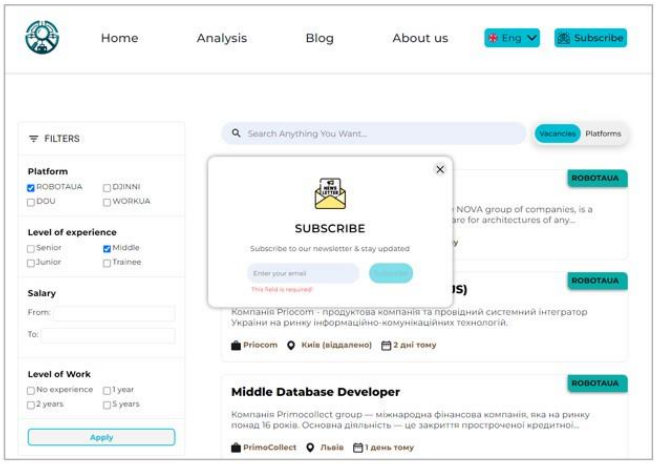
Головна сторінка



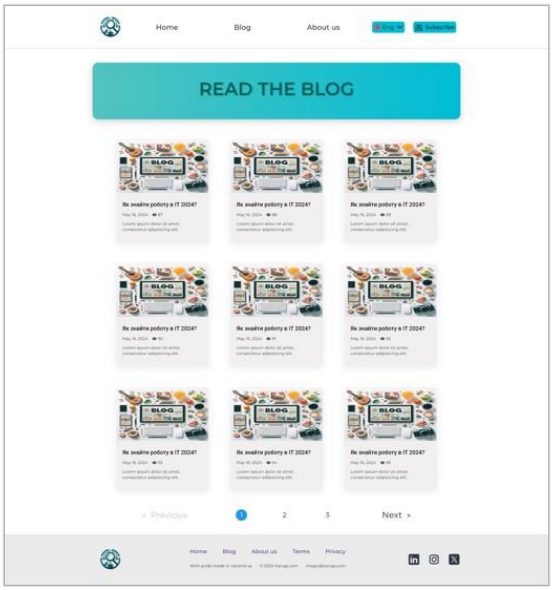
Головна сторінка з фільтрами та зміною мови

10

ЕКРАННІ ФОРМИ



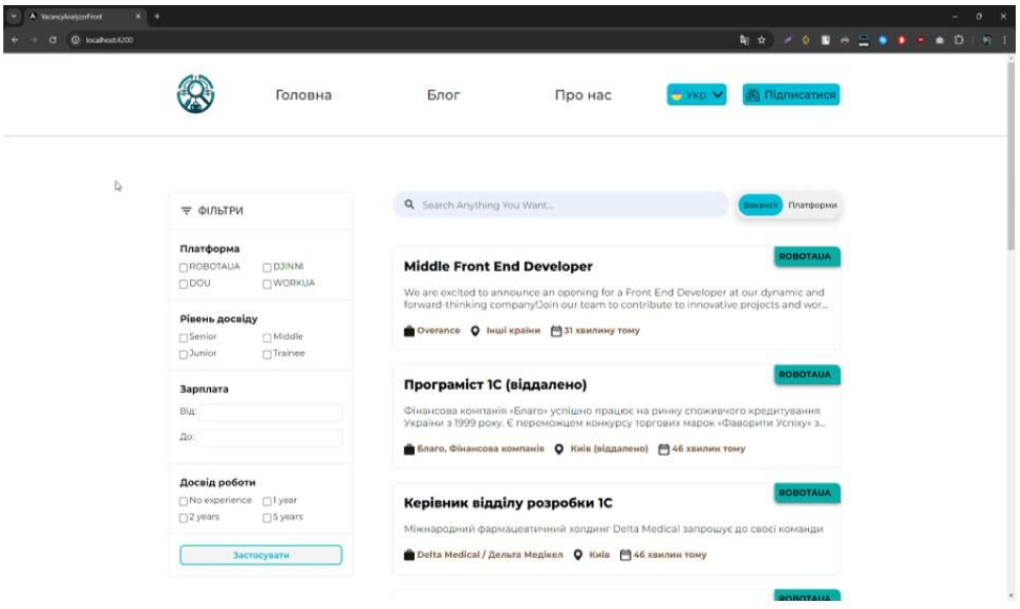
Форма підписки на новини



Блог

11

ДЕМОНСТРАЦІЯ РОБОТИ



12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Коноплястий А.Р. Розробка програмного забезпечення для аналізу вакансій з використанням фреймворку Spring Boot та Angular. Застосування програмного забезпечення в ІКТ: Матеріали Всеукраїнської науково-технічної конференції. Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024, Подано до друку.

2. Коноплястий А.Р. Розробка веб-застосунку для прогнозування технологічних тенденцій у галузі інформаційних технологій з використанням штучного інтелекту, Spring boot та Angular.
   Сучасні інтелектуальні інформаційні технології в науці та освіті: Матеріали IV Всеукраїнська науково-практична конференція. Збірник тез. 15.05.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024, Подано до друку.

ВИСНОВКИ

1. Досліджено ринок вакансій, дозволило ідентифікувати основні виклики, які включають швидку зміну технологічних трендів та вимог ринку праці, що є критичним для розробки адаптивного та ефективного програмного забезпечення.

2. Розроблено добре структурованої бази даних і чітко визначеної архітектури системи забезпечила ефективне взаємодію між компонентами, що значно підвищує продуктивність та масштабованість програмного рішення.

3. Систематичне модульне та інтеграційне тестування допомогло ідентифікувати та усунути помилки на ранніх етапах розробки, забезпечуючи надійність і стабільність фінального програмного продукту

ДОДАТОК Б ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.
0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.3</version>
    <relativePath/>
  </parent>
  <groupId>com.konolastiy</groupId>
  <artifactId>vacancy-analyzer</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>vacancy-analyzer</name>
  <description>vacancy-analyzer</description>

  <properties>
    <java.version>17</java.version>
    <spring-boot-dependencies.version>3.2.4</spring-
boot-dependencies.version>
    <lombock.version>1.18.30</lombock.version>
    <jackson-databind.version>2.16.1</jackson-
databind.version>

<mapstruct.version>1.5.5.Final</mapstruct.version>
    <lombok-mapstruct-
binding.version>0.2.0</lombok-mapstruct-
binding.version>
    <jsoup.version>1.17.2</jsoup.version>
    <selenium.version>4.20.0</selenium.version>
    <hibernate-envers.version>6.4.4.Final</hibernate-
envers.version>

<webdrivermanager.version>5.8.0</webdrivermanager.
version>
    <springdoc-openapi-starter-webmvc-
ui.version>2.5.0</springdoc-openapi-starter-webmvc-
ui.version>
    <h2.version>2.2.224</h2.version>
    <postgresql.version>42.7.2</postgresql.version>
    <mockito.version>5.10.0</mockito.version>
    <junit-jupiter.version>5.10.1</junit-jupiter.version>
    <junit.version>4.13.2</junit.version>
    <snakeyaml.version>2.0</snakeyaml.version>
  </properties>

  <!-- Dependency Management for Spring Boot -->
  <dependencyManagement>
    <dependencies>
      <dependency>
<groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-
dependencies</artifactId>
        <version>${spring-boot-
dependencies.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>

  <!-- Actual Dependencies -->
  <dependencies>
    <!-- Spring dependencies -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-
jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-
validation</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-mail</artifactId>
    </dependency>

    <!-- Project Lombok for reducing boilerplate code --
>
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
      <version>${lombock.version}</version>
    </dependency>
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>${jackson-databind.version}</version>
    </dependency>
    <dependency>
      <groupId>org.mapstruct</groupId>
      <artifactId>mapstruct</artifactId>
```

```
        <version>${mapstruct.version}</version>
      </dependency>
      <dependency>
        <groupId>org.mapstruct</groupId>
        <artifactId>mapstruct-processor</artifactId>
        <version>${mapstruct.version}</version>
      </dependency>
      <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok-mapstruct-
binding</artifactId>
        <version>${lombok-mapstruct-
binding.version}</version>
      </dependency>
      <dependency>
        <groupId>org.jsoup</groupId>
        <artifactId>jsoup</artifactId>
        <version>${jsoup.version}</version>
      </dependency>
      <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>${selenium.version}</version>
      </dependency>
      <dependency>
        <groupId>io.github.bonigarcia</groupId>
        <artifactId>webdrivermanager</artifactId>

<version>${webdrivermanager.version}</version>
      </dependency>
      <dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-starter-webmvc-
ui</artifactId>
        <version>${springdoc-openapi-starter-webmvc-
ui.version}</version>
      </dependency>
      <dependency>
        <groupId>org.hibernate.orm</groupId>
        <artifactId>hibernate-envers</artifactId>
        <version>${hibernate-envers.version}</version>
      </dependency>

      <!-- Database dependencies -->
      <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
        <version>${h2.version}</version>
      </dependency>
      <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
        <version>${postgresql.version}</version>
      </dependency>


      <!-- Test dependencies -->
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
      </dependency>
      <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-core</artifactId>
        <version>${mockito.version}</version>
        <scope>test</scope>
      </dependency>
      <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <scope>test</scope>
        <version>${junit-jupiter.version}</version>
      </dependency>
      <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <scope>test</scope>
        <version>${junit.version}</version>
      </dependency>
    </dependencies>

    <build>
      <finalName>vacancy-analyzer</finalName>

      <plugins>
        <!-- Excludes Lombok from Spring Boot build -->
        <plugin>

<groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-maven-
plugin</artifactId>
          <configuration>
            <excludes>
              <exclude>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
              </exclude>
            </excludes>
          </configuration>
        </plugin>

        <!-- Builds Docker image -->
        <plugin>
          <groupId>com.google.cloud.tools</groupId>
          <artifactId>jib-maven-plugin</artifactId>
          <version>3.4.0</version>
          <configuration>
            <to>

<image>kanap/${project.artifactId}</image>
            </to>
          </configuration>
        </plugin>

        <!-- Configures Maven Compiler Plugin with
```

```
MapStruct and Lombok for Spring Boot -->
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.10.1</version>
          <configuration>
            <source>${java.version}</source>
            <target>${java.version}</target>

<useIncrementalCompilation>true</useIncrementalCo
mpilation>
            <annotationProcessorPaths>
              <path>
                <groupId>org.mapstruct</groupId>
                <artifactId>mapstruct-
processor</artifactId>

<version>${mapstruct.version}</version>
              </path>
              <path>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <version>${lombok.version}</version>
              </path>
              <path>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok-mapstruct-
binding</artifactId>
                <version>${lombok-mapstruct-
binding.version}</version>
              </path>
            </annotationProcessorPaths>
            <compilerArgs>
              <compilerArg>
                -
Amapstruct.defaultComponentModel=spring
              </compilerArg>
            </compilerArgs>
          </configuration>
        </plugin>

      </plugins>
    </build>

</project>
```

```
@MappedSuperclass
@Getter
@Setter
@ToString
@EntityListeners(AuditingEntityListener.class)
public class BaseEntity {

    @CreatedDate
    @Column(updatable = false)
    private LocalDateTime createdAt;

    @CreatedBy
    @Column(updatable = false)
    private String createdBy;

    @LastModifiedDate
    @Column(insertable = false)
    private LocalDateTime updatedAt;

    @LastModifiedBy
    @Column(insertable = false)
    private String updatedBy;
}
```

```
@Entity
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Data
public class Blog extends BaseEntity {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

    private String title;

    private String content;

    private String img;

    private LocalDate datePublic;

    private Integer countViews;

}
```

```
@Entity
@AllArgsConstructor
@NoArgsConstructor
@Data
@Builder
public class NewsletterSubscriber extends BaseEntity {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

    private String email;
}
```

```
@Entity
@AllArgsConstructor
@NoArgsConstructor
@Data
@Builder
```

```
public class PlatformInfo {

   @Id
   @GeneratedValue(strategy =
GenerationType.IDENTITY)
   private Long id;

   private String platformName;
   private Integer seniorCount;
   private Integer middleCount;
   private Integer juniorCount;
   private Integer otherCount;
}


@Entity
@AllArgsConstructor
@NoArgsConstructor
@Data
@Builder
@JsonIgnoreProperties(value =
{"hibernateLazyInitializer", "handler"}, ignoreUnknown =
true)
public class Source {

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Integer id;

   @Enumerated(EnumType.STRING)
   private Platform name;

   private String link;
}


@Entity
@AllArgsConstructor
@NoArgsConstructor
@Data
@Builder
public class Vacancy extends BaseEntity {

   @Id
   @GeneratedValue(strategy =
GenerationType.IDENTITY)
   private Long id;

   private String vacancyName;
   private String cityName;

   private String date;

   private String companyName;

   @Length(max = 400)
   private String shortDescription;

   private String urlVacancy;

   private String salary;

   private String experienceLevel;

   private String englishLevel;

   private String programmingLanguage;

   @ManyToOne(fetch = FetchType.LAZY)
   @JoinColumn(name = "source_id")
   @NotAudited
   private Source source;

   @Column(name = "source_id", insertable = false,
updatable = false)
   private Integer source_id;
}

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
@Schema(
      name = "BlogResponse",
      description = "Schema to hold blog response
information"
)
public class BlogDto {

   @NotBlank
   @Schema(description = "Title of the blog", example =
"Sample Blog Title", required = true)
   private String title;

   @NotBlank
   @Schema(description = "Content of the blog",
example = "Sample Blog Content", required = true)
   private String content;

   @NotBlank
   @Schema(description = "Image of the blog", example
= "image.jpg", required = true)
   private String img;

   @NotNull
   @Schema(description = "Date when the blog was
published", example = "2024-04-30", required = true)
   private LocalDate datePublic;

   @NotNull
   @PositiveOrZero
   @Schema(description = "Number of views for the
blog", example = "100", required = true)
   private Integer countViews;
}
```

```
@Data
@AllArgsConstructor
@Schema(
      name = "ErrorResponse",
      description = "Schema to hold error response
information"
)
public class ErrorResponseDto {

   @Schema(
         description = "API path invoked by client"
   )
   private String apiPath;

   @Schema(
         description = "Error code representing the error
happened"
   )
   private HttpStatus errorCode;

   @Schema(
         description = "Error message representing the
error happened"
   )
   private String errorMessage;

   @Schema(
         description = "Time representing when the error
happened"
   )
   private LocalDateTime errorTime;

}
```

```
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
@Schema(
      name = "NewsletterSubscriberResponse",
      description = "Schema to hold NewsletterSubscriber
response information"
)
public class NewsletterSubscriberDto {

   @NotBlank
   @ValidEmail
   @Schema(description = "Email address of the
subscriber", example = "example@example.com")
   private String email;
}
```

```
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
@Schema(
      name = "PlatformInfoResponse",
      description = "Schema to hold platform response
information"
)
public class PlatformInfoDto {

   @NotBlank
   @Schema(description = "Name of the platform",
example = "DOU")
   private String namePlatform;

   @Nullable
   @Schema(description = "Number of senior level
vacancies", example = "10")
   private Integer senior;

   @Nullable
   @Schema(description = "Number of middle level
vacancies", example = "20")
   private Integer middle;

   @Nullable
   @Schema(description = "Number of junior level
vacancies", example = "30")
   private Integer junior;

   @Nullable
   @Schema(description = "Number of vacancies with
other experience levels", example = "5")
   private Integer others;
}
```

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@JsonIgnoreProperties({"hibernateLazyInitializer",
"handler"})
@Schema(
      name = "VacancyResponse",
      description = "Schema to hold vacancy response
information"
)
public class VacancyDto {

   @NotBlank(message = "Vacancy name must not be
blank")
   @Schema(description = "Name of the vacancy",
example = "Software Engineer")
   private String vacancyName;

   @NotBlank(message = "City name must not be
blank")
   @Schema(description = "City where the vacancy is
located", example = "New York")
   private String cityName;
```

```
    @NotBlank(message = "Date must not be blank")
    @Schema(description = "Date of the vacancy",
example = "Today")
    private String date;

    @NotBlank(message = "Company name must not be
blank")
    @Schema(description = "Name of the company
offering the vacancy", example = "Tech Corp")
    private String companyName;

    @NotBlank(message = "Short description must not be
blank")
    @Size(max = 400, message = "Short description must
be at most 400 characters")
    @Schema(description = "Short description of the
vacancy", example = "Join our team as a Software
Engineer!")
    private String shortDescription;

    @NotBlank(message = "Url vacancy must not be
blank")
    @Schema(description = "URL link to the vacancy",
example = "https://example.com/vacancies/9969521")
    private String urlVacancy;

    @NotBlank(message = "Salary must not be blank")
    @Schema(description = "Salary for the vacancy",
example = "2000 USD")
    private String salary;

    @Nullable
    @Schema(description = "Experience level required for
the vacancy", example = "Senior")
    private String experienceLevel;

    @Nullable
    @Schema(description = "English level required for the
vacancy", example = "Intermediate")
    private String englishLevel;

    @NotBlank(message = "Programming language must
not be blank")
    @Schema(description = "Programming language
required for the vacancy", example = "Java")
    private String programmingLanguage;

    @Schema(description = "Source of the vacancy",
implementation = Source.class)
    private Source sourceId;
}


public interface BlogRepository extends
JpaRepository<Blog, Long> {
}


public interface NewsletterSubscriberRepository
extends JpaRepository<NewsletterSubscriber, Long> {
    Optional<Object> findByEmail(String email);
}


public interface SourceRepository extends
JpaRepository<Source, Integer> {
}


public interface VacancyRepository extends
JpaRepository<Vacancy, Long>,
JpaSpecificationExecutor<Vacancy> {

    Integer countBySourceAndExperienceLevel(Source
source, String experienceLevel);
}


public class DjinniVacanciesCollector implements
Callable<List<Vacancy>> {

    private final Source source;
    private final String link;
    private final VacancyRepository vacancyRepository;
    private final VacancyMapper vacancyMapper;
    private final VacancyService vacancyService;

    public DjinniVacanciesCollector(Source source,
                                    String link,
                                    VacancyRepository
vacancyRepository,
                                    VacancyMapper vacancyMapper,
                                    VacancyService vacancyService) {
        this.source = source;
        this.link = link;
        this.vacancyRepository = vacancyRepository;
        this.vacancyMapper = vacancyMapper;
        this.vacancyService = vacancyService;
    }

    @Override
    public List<Vacancy> call() throws IOException {
        return collectVacancies();
    }

    private List<Vacancy> collectVacancies() throws
IOException {
        Document page = Jsoup.connect(link)
                .ignoreContentType(true)
                .referrer(GOOGLE_HOME_URL)
                .header("Accept-Encoding", "gzip")
                .userAgent("Mozilla/5.0 (Windows NT 6.1;
Win64; x64; rv:25.0) Gecko/20100101 Firefox/25.0")
                .maxBodySize(Integer.MAX_VALUE)
                .followRedirects(true)
                .timeout(25000)
                .get();
```

```
        Elements vacanciesOnPage = page.select(".list-
jobs__item.job-list__item");

        List<Vacancy> vacancies = new ArrayList<>();

        for (Element element : vacanciesOnPage) {
            VacancyDto vacancyDto = new VacancyDto();
            Element linkVacancy = element.select(".job-list-
item__link").first();

vacancyDto.setCompanyName(element.select("a.mr-
2").text().strip());

vacancyDto.setVacancyName(element.select("a.h3.job-
list-item__link").text().strip());
            vacancyDto.setDate(element.select(".mr-2
nobr").text());

vacancyDto.setCityName(element.select("span.location-
text").text().strip());

vacancyDto.setShortDescription(element.select("div.job
-list-item__description").text().strip());

vacancyDto.setUrlVacancy(DJINNI_HOME_PAGE_URL +
linkVacancy.attr("href"));

vacancyDto.setExperienceLevel(vacancyService.vacancy
SetExperience(vacancyDto));
            vacancyDto.setSourceId(source);

            Vacancy vacancy =
vacancyMapper.vacancyFromDto(vacancyDto);
            vacancy.setSource(source);
            vacancies.add(vacancy);
        }

        vacancyRepository.saveAll(vacancies);

        return vacancies;
    }
}


public class DouUaVacanciesCollector implements
Callable<List<Vacancy>> {

    private final Source source;
    private final String link;
    private final VacancyRepository vacancyRepository;
    private final VacancyMapper vacancyMapper;
    private final VacancyService vacancyService;

    public DouUaVacanciesCollector(Source source,
                                   String link,
                                   VacancyRepository
vacancyRepository,
                                   VacancyMapper vacancyMapper,
                                   VacancyService vacancyService) {
        this.source = source;
        this.link = link;
        this.vacancyRepository = vacancyRepository;
        this.vacancyMapper = vacancyMapper;
        this.vacancyService = vacancyService;
    }

    @Override
    public List<Vacancy> call() throws IOException {
        return collectVacancies();
    }

    private List<Vacancy> collectVacancies() throws
IOException {
        ArrayList<Vacancy> vacancies = new ArrayList<>();
        try {
            Document document =
Jsoup.connect(String.format(link))
                    .ignoreContentType(true)
                    .referrer(GOOGLE_HOME_URL)
                    .timeout(10000)
                    .header("Accept-Encoding", "gzip, deflate, br,
zstd")
                    .userAgent("Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/123.0.0.0 Safari/537.36")
                    .maxBodySize(Integer.MAX_VALUE)
                    .get();

            Elements vacancyElements = document.select(".l-
vacancy");

            for (Element vacancyElement : vacancyElements)
{
                VacancyDto vacancyDto = new VacancyDto();

vacancyDto.setUrlVacancy(vacancyElement.select("a.vt"
).attr("href"));

vacancyDto.setCityName(vacancyElement.select(".cities
").text());

vacancyDto.setCompanyName(vacancyElement.select("
a.company").text());

vacancyDto.setShortDescription(vacancyElement.select(
"div.sh-info").text());

vacancyDto.setDate(vacancyElement.select(".date").text
());
                String salary =
vacancyElement.select(".salary").text();
                vacancyDto.setSalary(!salary.isEmpty() ? salary
: "0");

vacancyDto.setVacancyName(vacancyElement.select("di
v.title > a.vt").first().text().strip());
```

```
vacancyDto.setExperienceLevel(vacancyService.vacancy
SetExperience(vacancyDto));
            vacancyDto.setSourceId(source);

            Vacancy vacancy =
vacancyMapper.vacancyFromDto(vacancyDto);
            vacancy.setSource(vacancyDto.getSourceId());
            vacancies.add(vacancy);
         }

         return vacancyRepository.saveAll(vacancies);

      } catch (IOException e) {
         throw new RuntimeException(e);
      }
   }

}


public class RobotaUaVacanciesCollector implements
Callable<List<Vacancy>> {

   private final Source source;
   private final String link;
   private final VacancyRepository vacancyRepository;
   private final VacancyMapper vacancyMapper;
   private final VacancyService vacancyService;

   public RobotaUaVacanciesCollector(Source source,
                        String link,
                        VacancyRepository
vacancyRepository,
                        VacancyMapper vacancyMapper,
                        VacancyService vacancyService) {
      this.source = source;
      this.link = link;
      this.vacancyRepository = vacancyRepository;
      this.vacancyMapper = vacancyMapper;
      this.vacancyService = vacancyService;
   }

   @Override
   public List<Vacancy> call() {
      return collectVacancies();
   }

   private List<Vacancy> collectVacancies() {
      List<Vacancy> vacancies = new ArrayList<>();
      System.setProperty(CHROME_DRIVER_NAME,
CHROME_DRIVER_PATH);
      ChromeOptions options = new ChromeOptions();
      options.addArguments("--headless");
      WebDriver driver = new ChromeDriver(options);
      WebDriver driverNextPage = new
ChromeDriver(options);

      try {
         driver.get(link);
         WebDriverWait wait = new
WebDriverWait(driver, Duration.ofSeconds(10));

wait.until(ExpectedConditions.presenceOfElementLocat
ed(By.className("card")));
         List<WebElement> cardElements =
driver.findElements(By.className("card"));
         for (WebElement cardElement : cardElements) {
            VacancyDto vacancyDto = new VacancyDto();

vacancyDto.setUrlVacancy(cardElement.getAttribute("h
ref"));

vacancyDto.setVacancyName(cardElement.findElement(
By.tagName("h2")).getText());
            List<WebElement> santamb10 =
cardElement.findElements(By.cssSelector("div.santa-flex
div.santa-mb-10:not(h2):not(.ng-star-inserted)"));
            if (santamb10.size() == 2) {

vacancyDto.setSalary(santamb10.get(0).getText().strip()
);

vacancyDto.setCityName(santamb10.get(1).getText().str
ip());
            } else {
               vacancyDto.setSalary("0");

vacancyDto.setCityName(santamb10.get(0).getText());
            }

vacancyDto.setCompanyName(cardElement.findElemen
t(By.className("santa-mr-20")).getText());

vacancyDto.setDate(cardElement.findElement(By.cssSel
ector(".santa-typo-secondary.santa-text-black-
500")).getText());

            if
(vacancyDto.getCityName().contains(vacancyDto.getCo
mpanyName())) {

vacancyDto.setCityName(vacancyDto.getCityName().rep
laceAll(vacancyDto.getCompanyName(), ""));
            }


driverNextPage.get(vacancyDto.getUrlVacancy());

vacancyDto.setShortDescription(getShortDescription(dri
verNextPage));


vacancyDto.setProgrammingLanguage(vacancyService.v
acancySetProgramingLanguage(vacancyDto));

vacancyDto.setExperienceLevel(vacancyService.vacancy
SetExperience(vacancyDto));
            vacancyDto.setSourceId(source);
```

```
            Vacancy vacancy =
vacancyMapper.vacancyFromDto(vacancyDto);
            vacancy.setSource(source);
            vacancies.add(vacancy);
         }

         vacancyRepository.saveAll(vacancies);

      } catch (Exception e) {
         System.out.println("Error collecting vacancies: " +
e.getMessage());
      } finally {
         if (driver != null) {
            driver.quit();
            driverNextPage.quit();
         }
      }
      return vacancies;
   }

   private String getShortDescription(WebDriver driver) {
      WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(10));
      List<WebElement> paragraphs =
wait.until(ExpectedConditions.presenceOfAllElementsLo
catedBy(By.cssSelector(".full-desc p")));

      for (WebElement paragraph : paragraphs) {
         String text = paragraph.getText().replace("\n",
"").strip();
         if (text.length() >= 50) {
            return text.substring(0, Math.min(text.length(),
200));
         }
      }
      return NO_DESCRIPTION_FOUND_MESSAGE;
   }

}


public class WorkUaVacanciesCollector implements
Callable<List<Vacancy>> {

   private final Source source;
   private final String link;
   private final VacancyRepository vacancyRepository;
   private final VacancyMapper vacancyMapper;
   private final VacancyService vacancyService;

   public WorkUaVacanciesCollector(Source source,
                        String link,
                        VacancyRepository
vacancyRepository,
                        VacancyMapper vacancyMapper,
                        VacancyService vacancyService) {
      this.source = source;
      this.link = link;
      this.vacancyRepository = vacancyRepository;
      this.vacancyMapper = vacancyMapper;
      this.vacancyService = vacancyService;
   }

   @Override
   public List<Vacancy> call() {
      return collectVacancies();
   }

   private List<Vacancy> collectVacancies() {
      return null;
   }
}


@Service
@RequiredArgsConstructor
public class BlogService {

   private final BlogRepository blogRepository;

   @Transactional
   public Blog savePost(@NonNull final Blog blog) {
      return blogRepository.save(blog);
   }

   @Transactional(readOnly = true)
   public Page<Blog> findAllBlogs(final Pageable
pageable) {
      return blogRepository.findAll(pageable);
   }
}


@Service
@Slf4j
@RequiredArgsConstructor
public class DjinniService {


   private final VacancyRepository vacancyRepository;
   private final SourceRepository sourceRepository;
   private final VacancyMapper vacancyMapper;
   private final VacancyService vacancyService;

   private static final Logger logger =
LoggerFactory.getLogger(DjinniService.class);

   @Async
   @Transactional
   public void getAllVacanciesDjinni() {
      Source source = sourceRepository.findById(2)
            .orElseThrow(() -> new
SourceNotFoundException(String.format(SOURCE_NOT_
FOUND_MESSAGE, 2)));

      String link = source.getLink();
```

```
        ExecutorService executorService =
Executors.newFixedThreadPool(THREAD_POOL_SIZE);
        List<Callable<List<Vacancy>>> tasks = new
ArrayList<>();
        for (int i = 1; i <= 40; i++) {
            tasks.add(new
WorkUaVacanciesCollector(source, link + i,
vacancyRepository, vacancyMapper, vacancyService));
        }

        try {
            List<Future<List<Vacancy>>> futures =
executorService.invokeAll(tasks);

            for (Future<List<Vacancy>> future : futures) {
                try {
                    future.get();
                } catch (Exception e) {
                    logger.warn(e.getMessage());
                }

            }
        } catch (InterruptedException e) {
            logger.warn("Thread interrupted while executing
tasks: " + e.getMessage());
        }
    }

}


@Slf4j
@Service
@RequiredArgsConstructor
public class DouService {

    private final VacancyRepository vacancyRepository;
    private final SourceRepository sourceRepository;
    private final VacancyMapper vacancyMapper;
    private final VacancyService vacancyService;

    private static final Logger logger =
LoggerFactory.getLogger(DouService.class);

    @Async
    @Transactional
    public void getAllVacanciesDouUa() {
        Source source = sourceRepository.findById(3)
                .orElseThrow(() -> new
SourceNotFoundException(String.format(SOURCE_NOT_
FOUND_MESSAGE, 2)));

        String link = source.getLink();

        ExecutorService executorService =
Executors.newFixedThreadPool(THREAD_POOL_SIZE);
        List<Callable<List<Vacancy>>> tasks = new
ArrayList<>();
        for (int i = 1; i <= 50; i++) {
            tasks.add(new DouUaVacanciesCollector(source,
                    link,
                    vacancyRepository,
                    vacancyMapper,
                    vacancyService));
        }

        try {
            List<Future<List<Vacancy>>> futures =
executorService.invokeAll(tasks);

            for (Future<List<Vacancy>> future : futures) {
                try {
                    future.get();
                } catch (Exception e) {
                    logger.warn(e.getMessage());
                }

            }
        } catch (InterruptedException e) {
            logger.warn("Thread interrupted while executing
tasks: " + e.getMessage());
        }
    }

}


@Service
@RequiredArgsConstructor
public class EmailService {

    @Value("${spring.mail.username}")
    private String from;
    private final MailSender mailSender;

    @Async
    @Transactional
    public void send(@NonNull final String to,
                     @NonNull final String subject,
                     @NonNull final String content
    ) {
        SimpleMailMessage message = new
SimpleMailMessage();
        message.setFrom(from);
        message.setTo(to);
        message.setSubject(subject);
        message.setText(content);

        mailSender.send(message);
    }
}


@Service
@RequiredArgsConstructor
public class JobCollectionService {

    private final DjinniService djinniService;
```

```
    private final DouService douService;
    private final RobotaUaService robotaUaService;
    private final WorkUaService workUaService;

    @Transactional
    public void fetchAllVacanciesFromAllPlatforms() {
        djinniService.getAllVacanciesDjinni();
        douService.getAllVacanciesDouUa();
        robotaUaService.getAllVacanciesRobotaUa();
        workUaService.getAllVacanciesWorkUa();
    }
}
```

```
@Service
@RequiredArgsConstructor
public class JsonFileDataService {

    private final ObjectMapper objectMapper;

    public Map<String, String[]>
loadJsonDataAsMap(String jsonFilePath) throws
IOException {
        return objectMapper.readValue(
                getClass().getResourceAsStream(jsonFilePath),
                new TypeReference<Map<String, String[]>>() {
                }
        );
    }
}
```

```
@Service
@RequiredArgsConstructor
public class NewsletterSubscriberService {

    private final NewsletterSubscriberRepository
newsletterSubscriberRepository;

    @Transactional
    public NewsletterSubscriber
saveNewsletterSubscriber(@NonNull final
NewsletterSubscriber subscriber) {
        final String email = subscriber.getEmail();

newsletterSubscriberRepository.findByEmail(email).ifPr
esent(s -> {
            throw new
SubscriberAlreadyExistsException(String.format(SUBSCR
IBER_ALREADY_EXITS_MESSAGE, email));
        });

        return
newsletterSubscriberRepository.save(subscriber);
    }
}
```

```
@Service
@RequiredArgsConstructor
public class PlatformInfoService {

    private final VacancyRepository vacancyRepository;
    private final SourceRepository sourceRepository;

    @Transactional
    public List<PlatformInfoDto>
getPlatformCountsVacancies() {
        List<Source> sources = sourceRepository.findAll();
        List<PlatformInfoDto> platformInfoList = new
ArrayList<>();

        for (Source source : sources) {
            PlatformInfoDto platformInfoDto = new
PlatformInfoDto();

platformInfoDto.setNamePlatform(source.getName().to
String());

            Integer seniorCount =
vacancyRepository.countBySourceAndExperienceLevel(s
ource, SENIOR);
            Integer middleCount =
vacancyRepository.countBySourceAndExperienceLevel(s
ource, MIDDLE);
            Integer juniorCount =
vacancyRepository.countBySourceAndExperienceLevel(s
ource, JUNIOR);
            Integer otherCount =
vacancyRepository.countBySourceAndExperienceLevel(s
ource, DEFAULT_EXPERIENCE_LEVEL);

            platformInfoDto.setSenior(seniorCount);
            platformInfoDto.setMiddle(middleCount);
            platformInfoDto.setJunior(juniorCount);
            platformInfoDto.setOthers(otherCount);

            platformInfoList.add(platformInfoDto);
        }

        return platformInfoList;
    }
}
```

```
@Service
@Slf4j
@RequiredArgsConstructor
public class RobotaUaService {

    private final VacancyRepository vacancyRepository;
    private final SourceRepository sourceRepository;
    private final VacancyMapper vacancyMapper;
    private final VacancyService vacancyService;

    private static final Logger logger =
LoggerFactory.getLogger(RobotaUaService.class);
```

```
  @Async
  @Transactional
  public void getAllVacanciesRobotaUa() {
    int page = 1;

    Source source = sourceRepository.findById(1)
        .orElseThrow(() -> new
SourceNotFoundException(String.format(SOURCE_NOT_
FOUND_MESSAGE, 1)));

    String link = source.getLink();

    ExecutorService executorService =
Executors.newFixedThreadPool(THREAD_POOL_SIZE);
    List<Callable<List<Vacancy>>> tasks = new
ArrayList<>();
    for (int i = 1; i <= 200; i++) {
      tasks.add(new
RobotaUaVacanciesCollector(source,
          link + i,
          vacancyRepository,
          vacancyMapper,
          vacancyService)
      );
    }
    try {
      List<Future<List<Vacancy>>> futures =
executorService.invokeAll(tasks);

      for (Future<List<Vacancy>> future : futures) {
        try {
          future.get();
        } catch (Exception e) {
          logger.warn(e.getMessage());
        }

      }
    } catch (InterruptedException e) {
      logger.warn("Thread interrupted while executing
tasks: " + e.getMessage());
    }
  }
}


@Service
@RequiredArgsConstructor
public class VacancyService {

  private final JsonFileDataService jsonFileDataService;
  private final VacancyRepository vacancyRepository;

  @Transactional
  public String vacancySetExperience(VacancyDto
vacancyDto) throws IOException {
    String title =
vacancyDto.getVacancyName().toLowerCase();

    Map<String, String[]> experienceLevels =
jsonFileDataService.loadJsonDataAsMap(EXPERIENCE_L
EVELS_JSON);

    return experienceLevels.entrySet().stream()
        .filter(entry -> Arrays.stream(entry.getValue())
            .anyMatch(title::contains))
        .map(Map.Entry::getKey)
        .findFirst()
        .orElse(DEFAULT_EXPERIENCE_LEVEL);
  }

  @Transactional
  public String
vacancySetProgramingLanguage(VacancyDto
vacancyDto) throws IOException {
    String title =
vacancyDto.getVacancyName().toLowerCase();

    Map<String, String[]> experienceLevels =
jsonFileDataService.loadJsonDataAsMap(PROGRAMMIN
G_LANGUAGE_JSON);

    return experienceLevels.entrySet().stream()
        .filter(entry -> Arrays.stream(entry.getValue())
            .anyMatch(title::contains))
        .map(Map.Entry::getKey)
        .findFirst()
        .orElse(DEFAULT_EXPERIENCE_LEVEL);
  }

  @Transactional(readOnly = true)
  public Page<Vacancy> findAllVacancies(final Pageable
pageable,
                        final String searchText,
                        final String experienceLevel,
                        final String platformName) {
    Specification<Vacancy> spec = Specification.where(

VacancySpecifications.hasTextInAttributes(searchText))

.and(VacancySpecifications.hasExperienceLevel(experie
nceLevel))

.and(VacancySpecifications.hasPlatformName(platform
Name));

    return vacancyRepository.findAll(spec, pageable);
  }
}


@Service
@Slf4j
@RequiredArgsConstructor
public class WorkUaService {

  private final VacancyRepository vacancyRepository;
  private final SourceRepository sourceRepository;
```

```
    private final VacancyMapper vacancyMapper;
    private final VacancyService vacancyService;

    private static final Logger logger =
LoggerFactory.getLogger(WorkUaService.class);

    @Async
    @Transactional
    public void getAllVacanciesWorkUa() {
        Source source = sourceRepository.findById(4)
                .orElseThrow(() -> new
SourceNotFoundException(String.format(SOURCE_NOT_
FOUND_MESSAGE, 4)));

        String link = source.getLink();

        ExecutorService executorService =
Executors.newFixedThreadPool(THREAD_POOL_SIZE);
        List<Callable<List<Vacancy>>> tasks = new
ArrayList<>();
        for (int i = 1; i <= 50; i++) {
            tasks.add(new DjinniVacanciesCollector(source,
                    link + i,
                    vacancyRepository,
                    vacancyMapper,
                    vacancyService));
        }

        try {
            List<Future<List<Vacancy>>> futures =
executorService.invokeAll(tasks);

            for (Future<List<Vacancy>> future : futures) {
                try {
                    future.get();
                } catch (Exception e) {
                    logger.warn(e.getMessage());
                }

            }
        } catch (InterruptedException e) {
            logger.warn("Thread interrupted while executing
tasks: " + e.getMessage());
        }
    }

}
```

```
@SpringBootApplication
@EnableAsync
@EnableJpaAuditing(auditorAwareRef =
"auditAwareImpl")
@OpenAPIDefinition(
        info = @Info(
                title = "Vacancy-analyzer REST API
Documentation",
                description = "Vacancy-analyzer REST API
Documentation",
                version = "v1",
                contact = @Contact(
                        name = "Andrew Konoplastiy",
                        email = "konoplastiy@gmail.com",
                        url = "https://www.vacancy-analyzer.com"
                ),
                license = @License(
                        name = "Apache 2.0",
                        url = "https://vacancy-analyzer.com"
                )
        ),
        externalDocs = @ExternalDocumentation(
                description = "Kanap-Vacancy-analyzer REST
API Documentation",
                url = "https://www.kanapbytes.com/swagger-
ui.html"
        )
)
public class VacancyAnalyzerApplication {

    public static void main(String[] args) {

SpringApplication.run(VacancyAnalyzerApplication.class,
args);
    }

}
```

```
@ControllerAdvice
public class GlobalExceptionHandler extends
ResponseEntityExceptionHandler {

    @Override
    protected ResponseEntity<Object>
handleMethodArgumentNotValid(
            MethodArgumentNotValidException ex,
HttpHeaders headers, HttpStatusCode status,
WebRequest request) {
        Map<String, String> validationErrors = new
HashMap<>();
        List<ObjectError> validationErrorList =
ex.getBindingResult().getAllErrors();

        validationErrorList.forEach((error) -> {
            String fieldName = ((FieldError) error).getField();
            String validationMsg =
error.getDefaultMessage();
            validationErrors.put(fieldName, validationMsg);
        });
        return new ResponseEntity<>(validationErrors,
HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<ErrorResponseDto>
handleGlobalException(Exception exception,
                                          WebRequest
webRequest) {
```

```
    ErrorResponseDto errorResponseDTO = new
ErrorResponseDto(
        webRequest.getDescription(false),
        HttpStatus.INTERNAL_SERVER_ERROR,
        exception.getMessage(),
        LocalDateTime.now()
    );
    return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_E
RROR)
        .body(errorResponseDTO);
  }


@ExceptionHandler(ResourceNotFoundException.class)
  public ResponseEntity<ErrorResponseDto>
handleResourceNotFoundException(ResourceNotFound
Exception exception,
                                WebRequest
webRequest) {
    ErrorResponseDto errorResponseDTO = new
ErrorResponseDto(
        webRequest.getDescription(false),
        HttpStatus.NOT_FOUND,
        exception.getMessage(),
        LocalDateTime.now()
    );
    return new ResponseEntity<>(errorResponseDTO,
HttpStatus.NOT_FOUND);
  }


@ExceptionHandler(VacancyAlreadyExistsException.clas
s)
  public ResponseEntity<ErrorResponseDto>
handleCardAlreadyExistsException(VacancyAlreadyExist
sException exception,
                                WebRequest
webRequest) {
    ErrorResponseDto errorResponseDTO = new
ErrorResponseDto(
        webRequest.getDescription(false),
        HttpStatus.BAD_REQUEST,
        exception.getMessage(),
        LocalDateTime.now()
    );
    return new ResponseEntity<>(errorResponseDTO,
HttpStatus.BAD_REQUEST);
  }

}


@Component("auditAwareImpl")
public class AuditAwareImpl implements
AuditorAware<String> {

  /**
   * Returns the current auditor of the application.
   *
   * @return the current auditor.
   */
  @Override
  public Optional<String> getCurrentAuditor() {
    return Optional.of("ADMIN_MS");
  }

}


@Configuration
public class ApplicationConfig {

  @Bean
  public RestTemplate restTemplate() {
    return new RestTemplate();
  }

  @Bean
  @Primary
  public ObjectMapper objectMapper() {
    ObjectMapper objectMapper = new
ObjectMapper();
    objectMapper.registerModule(new
JavaTimeModule());

objectMapper.disable(SerializationFeature.WRITE_DATE
S_AS_TIMESTAMPS);
    return objectMapper;
  }

  @Bean
  public  WebMvcConfigurer corsConfig() {
    return  new WebMvcConfigurer() {
      @Override
      public void addCorsMappings(CorsRegistry
registry) {
        registry.addMapping("/**")
            .allowedOrigins(FRONT_URL)
            .allowedMethods(HTTP_METHODS)
            .allowedHeaders("*");
      }
    };
  }

  @Bean
  public CorsConfigurationSource
corsConfigurationSource() {
    CorsConfiguration configuration = new
CorsConfiguration();
    UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();

configuration.setAllowedOrigins(Arrays.asList((FRONT_
URL)));

configuration.setAllowedMethods(Arrays.asList(HTTP_
METHODS));
```

```
configuration.setAllowedHeaders(Arrays.asList("*"));
        source.registerCorsConfiguration("/**",
configuration);
        return source;
    }

}


/**
 * Contains various constants used in the vacancy-
analyzer application.
 */
@UtilityClass
public class ApplicationConstants {

    /**
     * Constants related to URLs.
     */
    @UtilityClass
    public class UrlConstants {
        public static final String FRONT_URL =
"http://localhost:4200";
        public static final String GOOGLE_HOME_URL =
"https://www.google.com";
        public static final String DJINNI_HOME_PAGE_URL =
"https://djinni.co";
        public static final String CHROME_DRIVER_PATH =
"src/main/resources/chromedriver.exe";
        public static final String EXPERIENCE_LEVELS_JSON
= "/json/ExperienceLevels.json";
        public static final String
PROGRAMMING_LANGUAGE_JSON =
"/json/ProgrammingLanguages.json";

    }

    /**
     * Constants related to dates and formatting.
     */
    @UtilityClass
    public class Validation {
        public static final String DATE_PATTERN = "yyyy-
MM-dd HH:mm";

        /**
         * A regular expression for validating email
addresses.
         */
        public static final String EMAIL_REGEX = "^[a-zA-Z0-
9_!#$%&'*+/=?`{|}~^.-]+@[a-zA-Z0-9.-]+$";
        public static final Pattern EMAIL_PATTERN =
Pattern.compile(Validation.EMAIL_REGEX);
    }

    /**
     * Constants related to other configurations.
     */
    @UtilityClass
    public class ConfigConstants {
        public static final Integer THREAD_POOL_SIZE = 3;
        public static final String CHROME_DRIVER_NAME =
"webdriver.chrome.driver";
        public static final String[] HTTP_METHODS =
{"POST", "GET"};
    }

    /**
     * Constants related to error messages.
     */
    @UtilityClass
    public class ErrorMessageConstants {

        /**
         * Error message indicating that no description was
found for a vacancy.
         */
        public static final String
NO_DESCRIPTION_FOUND_MESSAGE = "Unfortunately,
no description was found for this vacancy.";
        public static final String
SOURCE_NOT_FOUND_MESSAGE = "Source with id %d
not found";
        public static final String
SUBSCRIBER_ALREADY_EXITS_MESSAGE = "The
subscriber already exists with email: %s";
    }

    @UtilityClass
    public class ExperienceLevelConstants {
        public static final String SENIOR = "Senior";
        public static final String MIDDLE = "Middle";
        public static final String JUNIOR = "Junior";
        public static final String
DEFAULT_EXPERIENCE_LEVEL = "Others";
    }

    @UtilityClass
    public class BlogControllerTest {
        public static final BlogDto TEST_BLOG_DTO = new
BlogDto();
        public static final Blog TEST_BLOG = new Blog();
    }

    @UtilityClass
    public class NewsletterSubscriberControllerTest {
        public static final NewsletterSubscriberDto
TEST_SUBSCRIBER_DTO = new
NewsletterSubscriberDto();
        public static final NewsletterSubscriber
TEST_SUBSCRIBER = new NewsletterSubscriber();
        public static final String ALREADY_ERROR_MES =
"Already subscribed";
        public static final String TEST_EMAIL =
"test@example.com";
    }
```

```
   @UtilityClass
   public class EmailServiceTest {
      public static final String TO = "test@example.com";
      public static final String SUBJECT = "Test Subject";
      public static final String CONTENT = "Hello, this is a
test email.";
   }

}


public class VacancySpecifications {

   public static Specification<Vacancy>
hasTextInAttributes(String searchText) {
      return (root, query, builder) -> {
         if (searchText == null || searchText.isEmpty())
return null;
         return builder.or(
               builder.like(root.get("vacancyName"), "%" +
searchText + "%"),
               builder.like(root.get("cityName"), "%" +
searchText + "%"),
               builder.like(root.get("companyName"), "%" +
searchText + "%"),
               builder.like(root.get("shortDescription"), "%"
+ searchText + "%"),
               builder.like(root.get("experienceLevel"), "%"
+ searchText + "%")
         );
      };
   }

   public static Specification<Vacancy>
hasExperienceLevel(String experienceLevel) {
      return (root, query, builder) ->
            experienceLevel == null ? null :
builder.equal(root.get("experienceLevel"),
experienceLevel);
   }

   public static Specification<Vacancy>
hasPlatformName(String platformName) {
      return (root, query, builder) -> {
         if (platformName == null) return null;
         Join<Vacancy, Source> sourceJoin =
root.join("source");
         return builder.equal(sourceJoin.get("name"),
platformName);
      };
   }
}


@Mapper(componentModel = "spring")
public interface BlogMapper {

   Blog blogFromDto(BlogDto blogDto);

   BlogDto blogToDto(Blog blog);
}


@Mapper(componentModel = "spring")
public interface NewsletterSubscriberMapper {

   NewsletterSubscriber
subscriberFromDto(NewsletterSubscriberDto
subscriberDto);

   NewsletterSubscriberDto
subscriberToDto(NewsletterSubscriber subscriber);
}


@Mapper(componentModel = "spring")
public interface VacancyMapper {

   @Mapping(source = "sourceId", target = "source")
   Vacancy vacancyFromDto(VacancyDto vacancyDto);

   @Mapping(source = "source", target = "sourceId")
   VacancyDto vacancyToDto(Vacancy vacancy);
}


@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends
RuntimeException {

   public ResourceNotFoundException(String
resourceName, String fieldName, String fieldValue) {
      super(String.format("%s not found with the given
input data %s : '%s'", resourceName, fieldName,
fieldValue));
   }
}


@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class SourceNotFoundException extends
RuntimeException {

   public SourceNotFoundException(String message) {
      super(message);
   }
}


@ResponseStatus(value = HttpStatus.BAD_REQUEST)
public class SubscriberAlreadyExistsException extends
RuntimeException {

   public SubscriberAlreadyExistsException(String
message) {
      super(message);
   }
}
```

```
@ResponseStatus(value = HttpStatus.BAD_REQUEST)
public class VacancyAlreadyExistsException extends
RuntimeException {

   public VacancyAlreadyExistsException(String message)
{
      super(message);
   }
}
```

```
version: "3"

services:
  backapp:
    image: kanap/vacancy-analyzer
    container_name: vacancy-analyzer
    environment:
      SPRING_PROFILES_ACTIVE: local
    ports:
      - 8081:8081
    depends_on:
      - selenium-hub
    volumes:
      - ./src/main/resources:/src/main/resources
    networks:
      - backend

  haproxy:
    image: haproxy:latest
    container_name: haproxy
    volumes:
      - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg
    ports:
      - 9000:9000
    depends_on:
      - backapp
    networks:
      - backend

  chrome:
    image: selenium/node-chrome:latest
    shm_size: 2gb
    depends_on:
      - selenium-hub
    environment:
      - SE_EVENT_BUS_HOST=selenium-hub
      - SE_EVENT_BUS_PUBLISH_PORT=4442
      - SE_EVENT_BUS_SUBSCRIBE_PORT=4443

  selenium-hub:
    image: selenium/hub:latest
    container_name: selenium-hub
    ports:
      - "4442:4442"
      - "4443:4443"
      - "4444:4444"
networks:
  backend:
    driver: "bridge"
```

```
global
    maxconn     4000

defaults
    mode http
    log global
    timeout connect 5000
    timeout client 50000
    timeout server 50000

frontend http-in
    bind 127.0.0.1:9000
    default_backend backapp

frontend stats_frontend
    bind *:9000
    stats enable
    stats show-legends
    stats uri /haproxy_stats
    stats refresh 10s

backend backapp
    balance roundrobin
    option forwardfor
    http-request set-header X-Forwarded-Port
%[dst_port]
    http-request add-header X-Forwarded-Proto https if {
ssl_fc }
    option httpchk GET /healthcheck HTTP/1.1
```

```
config.stopBubbling = true
lombok.addLombokGeneratedAnnotation = true
lombok.extern.findbugs.addSuppressFBWarnings = false
lombok.anyConstructor.addConstructorProperties =
true
```

```
@Component({
  selector: 'app-blog',
  templateUrl: './blog.component.html',
  styleUrls: ['./blog.component.scss']
})
export class BlogComponent implements OnInit {
  blogs: BlogItem[] = [];
  currentPage: number = 1;
  totalElements: number = 0;
  isEmpty: boolean = false;

  constructor(private blogService: BlogService) {}

  ngOnInit(): void {
    this.getAllBlogs();
```

```
  }

  onPageChange(page: number): void {
    this.currentPage = page;
    window.scrollTo({ top: 0, behavior: 'smooth' });
    this.getAllBlogs();
  }

  getAllBlogs(): void {
    this.blogService.getAllBlogs(this.currentPage - 1)
      .subscribe((data) => {
        this.blogs = data.content;
        this.totalElements = data.totalElements;
        this.isEmpty = this.blogs.length === 0;
      });
  }

}


@Component({
  selector: 'app-filter',
  templateUrl: './filter.component.html',
  styleUrls: ['./filter.component.scss']
})
export class FilterComponent implements OnInit {
  filterForm: FormGroup;
  platformName = ['ROBOTAUA', 'DJINNI', 'DOU',
'WORKUA'];
  experienceLevels = ['Senior', 'Middle', 'Junior',
'Trainee'];
  workLevels = ['No experience', '1 year', '2 years', '5
years'];

  constructor(private fb: FormBuilder, private
filterService: FilterService) {}

  ngOnInit(): void {
    this.initializeForm();
  }

  initializeForm() {
    this.filterForm = this.fb.group({
      platformName: new
FormArray(this.platformName.map(() =>
this.fb.control(false))),
      experienceLevel: new
FormArray(this.experienceLevels.map(() =>
this.fb.control(false))),
      salary: this.fb.group({
        from: '',
        to: ''
      }),
      work: new FormArray(this.workLevels.map(() =>
this.fb.control(false)))
    });
    console.log(this.filterForm)
  }

  applyFilters() {
    const filterData = this.filterForm.value;
    this.filterService.updateFilters({
      platformName:
filterData.platformName.map((checked, index) =>
checked ? this.platformName[index] : null).filter(value
=> value !== null),
      experienceLevel:
filterData.experienceLevel.map((checked, index) =>
checked ? this.experienceLevels[index] : null).filter(value
=> value !== null),
      salaryFrom: filterData.salary.from ?
parseInt(filterData.salary.from) : undefined,
      salaryTo: filterData.salary.to ?
parseInt(filterData.salary.to) : undefined,
      work: filterData.work.map((checked, index) =>
checked ? this.workLevels[index] : null).filter(value =>
value !== null)
    });
  }
}


@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.scss']
})
export class HeaderComponent implements OnInit {
  currentLanguage: string = 'eng';
  showLanguages = false;
  showSubscriptionModal = false;
  languages = [
    { value: 'ukr', name: 'Укр', icon: "assets/icons/ukraine-
flag.svg" },
    { value: 'eng', name: 'Eng', icon: "assets/icons/uk-
flag.svg" }
  ];
  form: FormGroup;

  constructor(public translate: TranslateService, private
subscriberService: SubscriberService, private snackBar:
MatSnackBar) {
  }

  ngOnInit(): void {
    this.initializeForm();
    this.initTranslate();
  }

  initTranslate() {
    this.translate.setDefaultLang('eng');
    this.translate.addLangs(this.languages.map(lang =>
lang.value));
    this.translate.use(this.currentLanguage);
  }

  toggleDropdown() {
    this.showLanguages = !this.showLanguages;
```

```
  }

  changeLanguage(language: string) {
   this.currentLanguage = language;
   this.showLanguages = false;
   this.translate.use(language);
  }

  get selectedLanguage() {
   return this.languages.find(lang => lang.value ===
this.currentLanguage);
  }

  get otherLanguages() {
   return this.languages.filter(lang => lang.value !==
this.currentLanguage);
  }

  openSubscriptionModal() {
   this.showSubscriptionModal = true;
  }

  closeSubscriptionModal() {
   this.showSubscriptionModal = false;
   this.form.reset();
  }

  subscribe(): void {
   if (this.form.valid) {
    const email: SubscriberModel =
this.form.getRawValue();


this.subscriberService.createNewsletterSubscriber(email
)
     .subscribe({
      next: () => {
       this.snackBar.open('Subscription successful',
'Close', { duration: 3000 });
       this.closeSubscriptionModal();
      },
      error: (error) => {
       this.snackBar.open('Failed to subscribe', 'Retry', {
duration: 3000 });
      }
     });
   }
  }

  initializeForm() {
   this.form = new FormGroup({
    email: new FormControl("", [Validators.required,
Validators.email, this.customeEmailValidator])
   });
  }

  getError(control): string {
   if (control.errors?.required && control.untouched)
    return 'This field is required!';
   else if (control.errors?.emailError &&
control.untouched)
    return 'Please enter valid email address!';
   else return '';
  }

  customeEmailValidator(control: AbstractControl) {
   const pattern = /^\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,20}$/;
   const value = control.value;
   if (!pattern.test(value) && control.untouched)
    return {
     emailError: true
    }
   else return null;
  }
}


@Component({
 selector: 'app-platform-card',
 templateUrl: './platform-card.component.html',
 styleUrls: ['./platform-card.component.scss']
})
export class PlatformCardComponent {
 @Input() platformInfo: PlatformInfoItem;

 constructor() {}
}


@Component({
 selector: 'app-search-overlay',
 templateUrl: './search-overlay.component.html',
 styleUrls: ['./search-overlay.component.scss']
})
export class SearchOverlayComponent {

 constructor(private searchBarService:
SearchBarService) {}

 recentSearches =
this.searchBarService.recentSearches;

 deleteRecentSearch(searchTerm: string) {

this.searchBarService.deleteRecentSearch(searchTerm);
 }

 performSearch(searchTerm: string) {
  this.searchBarService.search(searchTerm)
 }

 shouldDisplayRecentSearches() {
  return this.recentSearches.value !== null &&
this.recentSearches.value.length > 0;
 }

}
```

```
@Component({
  selector: 'app-vacancies',
  templateUrl: './vacancies.component.html',
  styleUrls: ['./vacancies.component.scss']
})
export class VacanciesComponent implements OnInit {

  overlayOpen = this.searchBarService.overlayOpen;
  searchTerm = this.searchBarService.searchTerm;
  vacancies: VacancyItem[] = [];
  currentPage: number = 1;
  totalElements: number = 0;
  isEmpty: boolean = false;
  platformsInfo: PlatformInfoItem[] = [];
  activeButton: string = 'vacancy';
  private filters: Filter = {platformName: [],
experienceLevel: [], work: []};
  private platform: BehaviorSubject<'vacancies' |
'platforms'> = new BehaviorSubject<'vacancies' |
'platforms'>('vacancies');


  constructor(private searchBarService:
SearchBarService,
              private vacancyService: VacancyService,
              private platformService: PlatformService,
              private filterService: FilterService) {
  }

  async ngOnInit(): Promise<void> {
    this.getAllVacancies();
    this.getAllPlatformsInfo();
    this.subscribeToFilters();
  }

  search(searchTerm: string) {
    if (!searchTerm) return;
    this.searchBarService.search(searchTerm);
    this.getAllVacancies(searchTerm);
  }

  openOverlay() {
    this.searchBarService.overlayOpen.next(true);
  }

  closeOverlay() {
    this.searchBarService.overlayOpen.next(false);
  }

  onPageChange(page: number): void {
    this.currentPage = page;
    window.scrollTo({top: 0, behavior: 'smooth'});
    this.getAllVacancies();
  }

  getAllVacancies(searchTerm: string = ''): void {
    this.vacancyService.getAllVacancies(this.filters,
searchTerm, this.currentPage - 1).subscribe((data) => {
      this.vacancies = data.content;
      this.totalElements = data.totalElements;
      this.isEmpty = this.vacancies.length === 0;
    });
  }

  getAllPlatformsInfo(): void {

this.platformService.getAllPlatformsInfo().subscribe((dat
a) => {
      this.platformsInfo = data;
    });
  }

  subscribeToFilters() {
    this.filterService.filter$
      .pipe(
        switchMap(filters => {
          this.filters = filters || {platformName: [],
experienceLevel: [], work: []};
          return
this.vacancyService.getAllVacancies(this.filters, "",
this.currentPage - 1);
        })
      )
      .subscribe(data => {
        this.vacancies = data.content;
        this.totalElements = data.totalElements;
        this.isEmpty = this.vacancies.length === 0;
      });
  }

  get platforms$() {
    return this.platform.asObservable();
  }

  flipToVacancies() {
    this.platform.next("vacancies");
    this.activeButton = 'vacancy';
  }

  flipToPlatforms() {
    this.platform.next("platforms");
    this.activeButton = 'platform';
  }

}


export interface BlogItem {
  title: string;
  content: string;
  img: string;
  datePublic: Date;
  countViews: number;
}
```

```
export interface Filter {
  platformName: string[];
  experienceLevel: string[];
  salaryFrom?: number;
  salaryTo?: number;
  work: string[];
}


export interface PlatformInfoItem {
  namePlatform: string;
  senior: number;
  middle: number;
  junior: number;
  others: number;
}


export interface SubscriberModel {
  email: string;
}


export interface VacancyItem {
  vacancyName: string;
  companyName: string;
  shortDescription: string;
  cityName: string;
  date: string;
  urlVacancy: string;
  sourceId: {
    name: string;
  };
}


@Injectable({
  providedIn: 'root'
})
export class BlogService {

  constructor(private httpClient: HttpClient) { }

  public getAllBlogs(page: number = 1, size: number = 9):
Observable<any> {
    return
this.httpClient.get(`${environment.apiUrl}/blogs?size=${
size}&page=${page}`);
  }
}


@Injectable({
  providedIn: 'root'
})
export class FilterService {
  private filterSubject = new BehaviorSubject<Filter |
null>(null);
  filter$ = this.filterSubject.asObservable();
  constructor() { }

  updateFilters(filters: Filter) {
    this.filterSubject.next(filters);
  }
}


@Injectable({
  providedIn: 'root'
})
export class PlatformService {

  constructor(private httpClient: HttpClient) { }

  public getAllPlatformsInfo(): Observable<any> {
    return
this.httpClient.get(`${environment.apiUrl}/platforms`);
  }
}


@Injectable({
  providedIn: 'root'
})
export class SearchBarService {

  overlayOpen = new BehaviorSubject<boolean>(false);
  recentSearches = new
BehaviorSubject<string[]>(JSON.parse(window.localStor
age.getItem('recentSearches') ?? '[]'));
  searchTerm = new BehaviorSubject<string>('');

  constructor() {}

  search(searchTerm: string) {
    this.searchTerm.next(searchTerm);
    this.overlayOpen.next(false);
    this.addToRecentSearches(searchTerm);
  }

  addToRecentSearches(searchTerm: string) {
    const lowerCaseTerm = searchTerm.toLowerCase();
    let updatedSearches = [
      lowerCaseTerm,
      ...this.recentSearches.value.filter((s) => s !==
lowerCaseTerm),
    ];
    updatedSearches = updatedSearches.slice(0, 5);
    this.recentSearches.next(updatedSearches);
    window.localStorage.setItem('recentSearches',
JSON.stringify(updatedSearches));
  }

  deleteRecentSearch(searchTerm: string) {
    const updatedSearches =
this.recentSearches.value.filter(s => s !== searchTerm);
    this.recentSearches.next(updatedSearches);
```

```
  }
}


@Injectable({
  providedIn: 'root'
})
export class SubscriberService {

  constructor(private httpClient: HttpClient) { }

  public createNewsletterSubscriber(subscriber: any):
Observable<any> {
    return
this.httpClient.post(`${environment.apiUrl}/subscribers`
, subscriber);
  }
}


@Injectable({
  providedIn: 'root'
})
export class VacancyService {
  constructor(private httpClient: HttpClient) {}

  public getAllVacancies(filters: Filter, searchText: string
= '', page: number = 1): Observable<any> {
    let params = new HttpParams()
      .set('searchText', searchText)
      .set('page', page.toString());

    if (filters.platformName &&
filters.platformName.length) {
      params = params.set('platformName',
filters.platformName.join(','));
    }
    if (filters.experienceLevel &&
filters.experienceLevel.length) {
      params = params.set('experienceLevel',
filters.experienceLevel.join(','));
    }
    if (filters.salaryFrom) {
      params = params.set('salaryFrom',
filters.salaryFrom.toString());
    }
    if (filters.salaryTo) {
      params = params.set('salaryTo',
filters.salaryTo.toString());
    }
    if (filters.work && filters.work.length) {
      params = params.set('work', filters.work.join(','));
    }

    return
this.httpClient.get(`${environment.apiUrl}/vacancies`, {
params });
  }
}
```