МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 29.00.00.000 ПЗ

Група ІПм-23-1

Дементєв Артем

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Дементєв Артем Олексійович

(прізвище, ім'я, по батькові)

УДК ___004.942_
(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі, методи та алгоритми побудови інноваційних платформ для пошуку роботи

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

А.О. Дементєв
(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Козак Олексій Федорович, к.т.н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту
Завідувач кафедри
доц. Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль
доц. Вовк Р.Б.
(посада) (підпис) (дата) (ініціали та прізвище)

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

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Дементєву Артему Олексійовичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи **“**Моделі, методи та алгоритми побудови інноваційних платформ для пошуку роботи**”**
керівник проєкту (роботи) Козак Олексій Федорович, к.т.н.
затверджені наказом закладу вищої освіти від **“** 22 ” листопада 2024 р. № 781/7

2. Строк подання студентом проєкту (роботи) 12 грудня 2024 р.

3. Вихідні дані до проєкту (роботи) Архітектура, формальний опис та алгоритми функціонування систем пошуку роботи

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Теоретичні основи створення інноваційних платформ для пошуку роботи
2. Сучасні технології та інструменти для побудови платформ працевлаштування
3. Розробка системи комунікації між користувачами та компаніями
4. Безпека та збереження конфіденційності даних користувачів

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Схема роботи алгоритму шифрування DES (рис. 2.1, ст. 45)
2. Схема роботи алгоритм3 шифрування AES (рис. 2.2, ст. 46)
3. Схематичне зображення діаграми аналітики новостворених вакансій та їх перегляд користувачами (рис. 3.10, ст. 73)
4. UML діаграма послідовності алгоритму розміщення резюме (рис. 3.11, ст. 74)
5. UML діаграма послідовності алгоритму розміщення вакансій (рис. 3.12, ст. 76)
6. UML діаграма послідовності алгоритму створення сповіщень (рис. 3.13, ст. 78)

6. Консультанти розділів проєкту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц. к.т.н. Вовк Р. Б.	

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури за темою дослідження	10.09.2024	виконано
2	Аналіз існуючих моделей та платформ для пошуку роботи та їх алгоритмів	20.09.2024	виконано
3	Формулювання вимог до побудови інноваційної платформи для пошуку роботи	10.10.20234	виконано
4	Розробка концепції архітектури інноваційної платформи	23.10.2024	виконано
5	Програмна реалізація модулів інноваційної платформи	1.11.2024	виконано
6	Тестування та оцінка ефективності розробленого рішення	1.12.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 95 с., 30 рис., 38 джерел.

Тема: моделі, методи та алгоритми побудови інноваційних платформ для пошуку роботи.

Об'єкт дослідження: інноваційні платформи для пошуку роботи.

Мета роботи: розробка моделей, методів та алгоритмів, які підвищують ефективність і доступність інноваційних платформ для пошуку роботи.

Предмет дослідження: методологія побудови, алгоритми функціонування та оптимізації інноваційних платформ для пошуку роботи.

Результати дослідження:

- проведено аналіз існуючих платформ для пошуку роботи, визначено їх переваги та недоліки;
- розроблено модель архітектури інноваційної платформи, яка враховує сучасні тенденції автоматизації та штучного інтелекту;
- запропоновано алгоритми, що забезпечують персоналізований підхід до пошуку роботи для користувачів;
- реалізовано прототип платформи, що дозволяє підвищити ефективність співпраці між роботодавцями та шукачами роботи.

Висновок:

Розроблені моделі, методи та алгоритми які можуть слугувати основою для створення новітньої платформи для пошуку роботи. Вони сприяють автоматизації процесів пошуку вакансій, що дозволяє забезпечити ефективну взаємодію між роботодавцями та здобувачами, а також підвищити якість та швидкість працевлаштування.

ПОШУК РОБОТИ, АНАЛІТИКА РИНКУ ПРАЦІ, ВЕБ-ПЛАТФОРМА, МОБІЛЬНИЙ ДОДАТОК, ІНТЕГРАЦІЯ З СОЦІАЛЬНИМИ МЕРЕЖАМИ, ХМАРНЕ ЗБЕРІГАННЯ ДАНИХ, БЕЗПЕКА ТА ПРИВАТНІСТЬ ДАНИХ, АНКЕТУВАННЯ ТА СКЛАДАННЯ РЕЗЮМЕ, ОБРАЗ РЕЗЮМЕ ЗА ШАБЛОНОМ.

ANNOTATION

Master's work: 95 p., 30 fig., 38 sources.

Topic: models, methods, and algorithms for building innovative job search platforms.

Object of research: innovative job search platforms.

Purpose: to develop models, methods, and algorithms that enhance the efficiency and accessibility of innovative job search platforms.

Subject of research: the methodology of building, algorithms for functioning, and optimization of innovative job search platforms.

Research results:

- an analysis of existing job search platforms has been conducted, identifying their strengths and weaknesses;
- a model of the architecture for an innovative job search platform has been developed, incorporating modern trends in automation and artificial intelligence;
- algorithms providing a personalized approach to job searching for users have been proposed;
- a prototype platform has been implemented, enabling more effective collaboration between employers and job seekers.

Conclusion:

The developed models, methods, and algorithms can serve as a foundation for creating a cutting-edge job search platform. They facilitate the automation of vacancy search processes, enabling efficient interaction between employers and job seekers while improving the quality and speed of employment.

JOB SEARCH, LABOR MARKET ANALYTICS, WEB PLATFORM, MOBILE APPLICATION, INTEGRATION WITH SOCIAL NETWORKS, CLOUD DATA STORAGE, DATA SECURITY AND PRIVACY, SURVEY AND RESUME BUILDING, TEMPLATE-BASED RESUME DESIGN.

ЗМІСТ

Стр.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення,

СУБД – система управління базами даних,

API (Application Program Interface) – прикладний програмний інтерфейс,

HTTPS – протокол безпечної передачі даних,

VS (Visual Studio) – програмне середовище для написання backend логіки,

VS Code (Visual Studio Code) – програмне середовище для написання fronted логіки,

Azure – платформа хмарних обчислень з набором служб, що постійно розширюється,

MVC (Model/View/Controller) – архітектурний шаблон проєктування,

DTO (Data Transfer Object) – шаблон проєктування, який використовують для передачі даних між підсистемами програми,

DI (Dependency Injection) – підхід, при якому об'єкт отримує свої залежності (інші об'єкти) зовні, а не створює їх самостійно.

EF (Entity Framework) – структура об'єктно-реляційного відображення (ORM) з відкритим кодом для ADO.NET

ВСТУП

Сучасний ринок праці зазнає значних трансформацій, зумовлених глобалізацією, цифровізацією та розвитком інформаційних технологій. Постійне збільшення кількості вакансій, зміна форматів роботи, таких як віддалена зайнятість та фріланс, створюють нові виклики як для шукачів роботи, так і для роботодавців. У цьому контексті виникає необхідність створення інноваційних платформ, які б забезпечували ефективний пошук роботи, адаптивний до сучасних умов [6].

Наукова проблема полягає у розробці моделей, методів і алгоритмів, які дозволяють підвищити точність підбору вакансій, прискорити процес працевлаштування та забезпечити зручність використання таких платформ для різних категорій користувачів. Актуальність дослідження зумовлена необхідністю інтеграції сучасних технологій, таких як машинне навчання, хмарні сервіси та штучний інтелект, у процес побудови платформ для пошуку роботи.

Мета роботи полягає в аналізі існуючих рішень, розробці нових методологій та їхньому впровадженні для створення інноваційних платформ, які б відповідали потребам сучасного ринку праці. У рамках дослідження буде розглянуто поточний стан проблеми, визначено вихідні дані для побудови ефективних рішень і обґрунтовано необхідність розробки нових підходів.

Робота охоплює такі аспекти: аналіз існуючих платформ і технологій у сфері пошуку роботи, розробка моделей і алгоритмів для покращення точності підбору вакансій, а також тестування і впровадження створених рішень. Значущість дослідження полягає у сприянні розвитку ринку праці через інноваційні технології, що відповідають вимогам сучасних користувачів.

Актуальність роботи

У сучасному суспільстві ринок праці є важливим елементом економічної системи, який безпосередньо впливає на соціальну стабільність та добробут громадян [2]. З огляду на стрімкий розвиток технологій, виникає необхідність у створенні нових рішень, здатних ефективно задовольнити запити користувачів. Застарілі методи пошуку роботи та підбору персоналу не відповідають сучасним викликам, таким як

збільшення обсягів даних, зростання конкуренції серед кандидатів та зміна форматів праці.

Впровадження інноваційних платформ для пошуку роботи є нагальною потребою. Такі платформи мають вирішувати ключові проблеми, зокрема:

- автоматизацію процесу підбору вакансій,
- адаптацію до вимог ринку праці,
- забезпечення доступності актуальної інформації для різних категорій користувачів.

Особливої актуальності набуває використання технологій штучного інтелекту, машинного навчання та аналітики великих даних для побудови платформ, що можуть забезпечити високу точність підбору вакансій та кандидатів. Крім того, враховуючи розвиток гібридної та віддаленої роботи, необхідність інтеграції функціоналу для підтримки цих форматів стає критично важливою.

Також важливим аспектом є персоналізація користувацького досвіду, яка передбачає не лише автоматичне підбирання вакансій на основі навичок та уподобань кандидата, але й надання рекомендацій для розвитку кар’єри. Таким чином, створення сучасних платформ здатне не лише спростити пошук роботи, але й сприяти професійному зростанню користувачів, підвищуючи їхню конкурентоспроможність на ринку праці [4].

Отже, дослідження у цій сфері є актуальним і перспективним, оскільки сприяє вирішенню низки суспільно важливих завдань, серед яких підвищення ефективності працевлаштування, покращення взаємодії між роботодавцями та кандидатами та забезпечення доступу до сучасних інструментів кар’єрного розвитку.

Порівняння роботи з відомими розв’язаннями проблеми

Дослідження за темою магістерської роботи тісно пов’язане з науковими напрямками кафедри комп'ютерних наук та інформаційних технологій, яка орієнтована на розробку інноваційних рішень у сфері цифрових платформ та їх застосування в різних галузях економіки, зокрема в сфері працевлаштування. Цей напрямок підтримує розвиток сучасних інформаційних систем, здатних вирішувати

актуальні проблеми ринку праці, зокрема через автоматизацію та персоналізацію процесів пошуку роботи.

Зокрема, тематика дослідження відповідає науковим планам університету, спрямованим на інноваційний розвиток у сфері IT-технологій, цифровізації економічних та соціальних процесів, а також покращення взаємодії між роботодавцями та кандидатами на роботу. Оскільки університет активно підтримує інтеграцію передових технологій у навчальний процес, дослідження має важливе значення для розширення наукового потенціалу кафедри та університету загалом.

Дослідження також узгоджується з галузевими та державними програмами, що спрямовані на цифрову трансформацію ринку праці, як-от програми розвитку національних цифрових платформ, які покликані модернізувати процеси працевлаштування, покращити доступ до актуальних вакансій та підвищити ефективність кадрових рішень. Важливість цієї роботи полягає в її внеску в реалізацію національних ініціатив, спрямованих на покращення умов для працевлаштування, розвитку гнучких форм зайнятості та впровадження новітніх технологій в управління людськими ресурсами.

Таким чином, робота має безпосередній зв'язок із пріоритетами наукової діяльності кафедри, університету, а також з актуальними державними ініціативами, що сприяють розвитку інновацій у сфері цифрових технологій на ринку праці.

Мета і задачі дослідження

Мета магістерської роботи полягає у розробці інноваційної платформи для пошуку роботи, що забезпечить ефективну та зручну взаємодію між кандидатами на роботу та роботодавцями, а також автоматизує процес підбору вакансій, що найбільше відповідають кваліфікації та вимогам кандидатів. Створення такої платформи передбачає впровадження сучасних технологій, зокрема методів машинного навчання, аналізу великих даних і штучного інтелекту, для надання точних і персоналізованих рекомендацій для кожного користувача. Крім того, передбачається інтеграція платформи з соціальними мережами та іншими популярними онлайн-ресурсами для розширення її функціональних можливостей і покращення доступу до різноманітних вакансій та кандидатів.

Досягнення цієї мети потребує вирішення низки важливих завдань, що зумовлені сучасними вимогами ринку праці, швидким розвитком цифрових технологій та високими вимогами до зручності та безпеки користування платформами. Враховуючи це, для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Аналіз існуючих платформ для пошуку роботи: для ефективної розробки нової платформи важливо вивчити існуючі рішення в цій галузі, проаналізувати їхні переваги та недоліки, а також виявити інноваційні функції, які можуть бути реалізовані в новому продукті. Це дозволить зрозуміти поточний стан ринку та виявити основні вимоги користувачів.

2. Розробка концепції інтерфейсу та архітектури платформи: важливим етапом є створення інтерфейсу, який буде зручним та інтуїтивно зрозумілим для користувачів, а також архітектури платформи, яка дозволить швидко та безпечно обробляти великі обсяги даних, що забезпечить швидкість та ефективність роботи системи. Архітектура повинна враховувати вимоги до масштабованості і здатність системи працювати в умовах високих навантажень.

3. Розробка та впровадження алгоритмів машинного навчання: одним із ключових аспектів цієї роботи є застосування алгоритмів машинного навчання для покращення точності підбору вакансій та кандидатів. Алгоритми мають бути здатні враховувати історичні дані користувачів, їх професійний досвід, освіту та інші параметри для створення персоналізованих рекомендацій. Важливо також враховувати динамічні зміни на ринку праці та швидко адаптувати алгоритми під нові вимоги.

4. Оцінка ефективності запропонованої платформи: після розробки базових функціональних можливостей платформи необхідно провести її тестування на реальних користувачах, що дозволить оцінити її ефективність і зручність. Це дозволить виявити можливі недоліки та зробити необхідні коригування перед масовим впровадженням платформи на ринок.

5. Розробка механізмів інтеграції з соціальними мережами та іншими платформами: інтеграція з популярними соціальними мережами (Facebook, LinkedIn,

Telegram) та іншими онлайн-ресурсами, такими як платформи для онлайн-освіти чи професійного розвитку, дозволить значно розширити функціональні можливості платформи та залучити більше користувачів. Це дасть змогу кандидатам отримувати більше вакансій, а роботодавцям – мати доступ до більшої кількості кваліфікованих кандидатів.

6. Забезпечення безпеки та конфіденційності даних: оскільки платформи для пошуку роботи обробляють чутливі особисті дані, таких як резюме, контактні дані та професійна історія користувачів, важливо впровадити високі стандарти безпеки. Це включає захист від несанкціонованого доступу до даних, їх шифрування, а також дотримання всіх норм і вимог щодо захисту персональних даних, встановлених законодавством.

7. Розробка стратегії впровадження та масштабування платформи: після розробки та тестування платформи необхідно розробити стратегію її впровадження на ринку. Це включає вибір каналів для залучення користувачів, стратегії маркетингу, а також план масштабування платформи, щоб вона могла обслуговувати великий потік користувачів та зростаючі вимоги ринку праці.

Кожен з цих етапів є важливим для досягнення поставленої мети – створення високотехнологічної платформи для пошуку роботи, що буде відповідати вимогам як кандидатів, так і роботодавців, підвищить ефективність процесу працевлаштування та сприятиме розвитку ринку праці в цілому.

Об’єктом дослідження є інноваційні платформи для пошуку роботи, що включають в себе механізми автоматизованого підбору вакансій, персоналізовані рекомендації для кандидатів, а також інтерфейси для взаємодії з роботодавцями. Це також стосується технологій, що лежать в основі таких платформ, зокрема методів машинного навчання, аналізу даних та інтеграцій з соціальними мережами. Дослідження охоплює всі аспекти, пов’язані з розробкою та впровадженням таких платформ, починаючи від архітектури системи до забезпечення безпеки даних і взаємодії користувачів.

Предметом дослідження є методи, алгоритми та технології, що використовуються для створення інноваційних платформ для пошуку роботи. Це включає в себе:

Розробку алгоритмів машинного навчання для аналізу даних та персоналізації рекомендацій для кандидатів і роботодавців.

- Моделі архітектури платформ, включаючи інтеграцію з соціальними мережами та іншими онлайн-ресурсами.
- Технології забезпечення безпеки та конфіденційності даних користувачів на таких платформах.
- Оцінку ефективності та зручності використання платформ для різних категорій користувачів.
- Методи інтеграції з іншими онлайн-платформами, такими як освітні ресурси та соціальні мережі, для розширення можливостей платформи.
- Методи дослідження в цій роботі спрямовані на розробку інноваційної платформи для пошуку роботи, а також на дослідження ефективності її функціональних можливостей та оптимізації процесу підбору вакансій та кандидатів. Для досягнення поставленої мети були застосовані наступні методи:
- Метод системного аналізу – використовувався для вивчення загальної архітектури платформи, її складових частин і взаємодії між ними. Цей метод дозволив чітко визначити вимоги до платформи, що включають функціональні можливості, безпеку даних, інтеграцію з соціальними мережами та ефективність взаємодії з користувачами.
- Метод машинного навчання – застосовувався для розробки алгоритмів персоналізованих рекомендацій, зокрема для підбору вакансій, що відповідають інтересам і кваліфікації кандидата, а також для автоматизації оцінки кандидатів за допомогою моделей класифікації та регресії. Цей метод дозволив реалізувати інтелектуальний підхід до підбору вакансій і кандидатів.
- Метод моделювання – використано для створення прототипів інтерфейсу користувача платформи. Моделювання дозволило тестувати й оптимізувати

взаємодію користувача з платформою на різних етапах розробки, а також оцінити зручність використання.

- Метод аналізу великих даних (Big Data) – застосовувався для обробки та аналізу великого обсягу даних з соціальних мереж, професійних сайтів та інших платформ. Цей метод допоміг зібрати інформацію для покращення алгоритмів персоналізації та створення точних рекомендацій для користувачів.
- Метод структурного моделювання – використовувався для побудови структури бази даних платформи та забезпечення її ефективного функціонування. Це включає визначення та моделювання зв'язків між різними елементами системи, зокрема між вакансіями, кандидатами та роботодавцями.
- Метод експерименту – застосовувався для оцінки ефективності роботи платформи на реальних користувачах. Дослідження проводилось шляхом тестування платформи, збору зворотного зв'язку від користувачів і аналізу результатів взаємодії з системою для подальшого вдосконалення інтерфейсу та алгоритмів.
- Метод анкетування та інтерв'ювання – використовувався для збору інформації від кінцевих користувачів платформи (кандидатів та роботодавців). Це дозволило визначити потреби та уподобання користувачів, а також вдосконалити функціональність платформи, забезпечивши відповідність вимогам ринку праці.
- Методи тестування програмного забезпечення – застосовувалися для перевірки коректності роботи платформи та оцінки її стабільності під високими навантаженнями. Тестування дозволило виявити можливі баги, збої або неефективні алгоритми і виправити їх до фінальної версії.

Кожен з методів дослідження був використаний для вирішення конкретних завдань, пов'язаних з розробкою, оцінкою і вдосконаленням інноваційної платформи для пошуку роботи, забезпечуючи її ефективність і зручність для кінцевих користувачів.

Наукова новизна одержаних результатів

Наукова новизна цієї роботи полягає в наступних аспектах:

- Вперше розроблено комплексний підхід до створення інноваційної платформи для пошуку роботи, що поєднує технології машинного навчання, великих

даних і системного аналізу для персоналізованого підбору вакансій та кандидатів. Запропоновано новий метод оцінки ефективності роботи таких платформ на основі інтеграції з соціальними мережами та аналітики даних.

- Удосконалено алгоритм рекомендацій, що використовує машинне навчання для прогнозування та підбору найбільш підходящих вакансій на основі профілю користувача. Цей підхід дозволяє значно підвищити точність і швидкість підбору вакансій, адаптуючи платформу до індивідуальних потреб користувача.
- Дістала подальший розвиток концепція інтеграції платформи з соціальними мережами та іншими онлайн-ресурсами, що дає змогу забезпечити актуальність і достовірність даних про кандидата та вакансії. Це дозволяє автоматично оновлювати профілі користувачів і пропонувати вакансії, які відповідають не лише професійним навичкам, але й соціальним інтересам.
- Вперше застосовано методи аналізу великих даних для оцінки ринку праці в реальному часі, що дозволяє передбачати зміни в тенденціях трудової міграції, попит на певні професії та кваліфікації. Це дозволяє кандидатам своєчасно коригувати свої навички, а роботодавцям — адаптувати стратегії найму до нових умов.
- Розроблено новий підхід до побудови бази даних для платформи, який оптимізує обробку та збереження великої кількості структурованих та неструктурованих даних**.** Це підвищує швидкість доступу до інформації та знижує навантаження на сервери під час високих пікових навантажень.
- Удосконалено методи тестування інтерфейсу користувача для онлайн-платформ, що дозволяє оптимізувати взаємодію з користувачами та забезпечити зручність використання при великих обсягах даних і високій динаміці змін. Розроблена система тестування ефективності взаємодії користувачів з платформою на основі збору даних про їхні вподобання та дії, що дозволяє постійно покращувати дизайн та функціональність.

Таким чином, результати дослідження не лише розширюють теоретичні засади побудови інноваційних платформ для пошуку роботи, але й пропонують практичні рішення для їх впровадження в реальних умовах ринку праці.

Практичне значення одержаних результатів

Одержані результати мають значну практичну цінність для розвитку платформ для пошуку роботи, зокрема в аспектах удосконалення алгоритмів рекомендацій, інтеграції з соціальними мережами та аналізу великих даних.

Запропоновані методи дозволяють створити ефективнішу систему підбору вакансій та кандидатів, що значно покращує процес працевлаштування. Вони можуть бути впроваджені в роботі рекрутингових компаній, онлайн-платформ для пошуку роботи, а також в інші організації, які займаються аналізом ринку праці.

Рекомендації щодо використання розроблених методів можуть бути застосовані для оптимізації інтерфейсів користувачів та підвищення ефективності їх взаємодії з платформами.

В результаті реалізації цих підходів можна очікувати значне зниження часу на підбір вакансій, підвищення якості відповідності між кандидатами та роботодавцями, а також поліпшення загального досвіду користувачів платформ. На даному етапі результати дослідження готові до практичного застосування та можуть бути впроваджені в різних організаціях, що займаються онлайн-рекрутингом та аналізом даних.

Структура магістерської роботи.

Магістерська робота викладена на 95 сторінках друкованого тексту, який складається з вступу, трьох розділів, висновків, списку використаних джерел (38 найменувань). Робота містить 30 рисунків.

РОЗДІЛ 1 ОГЛЯД СУЧАСНИХ ТЕХНОЛОГІЙ РОЗРОБКИ ЗАСТОСУНКІВ

1.1 Основи розробки на платформі ASP.NET Core

1.1.1 Особливості ASP.NET Core та його можливості для створення масштабованих додатків

ASP.NET Core – це кросплатформовий, високопродуктивний фреймворк для створення сучасних хмарних, веб і мобільних додатків [37]. Він має низку особливостей та можливостей, що роблять його потужним інструментом для розробки масштабованих додатків. Він є частиною екосистеми .NET і підтримує як розробку бекенд-додатків, так і інтеграцію з клієнтськими технологіями, такими як Angular, React і Blazor.

Використання ASP.NET Core:

- Веб-додатки: створення сайтів і веб-інтерфейсів.
- Веб-сервіси: реалізація RESTful API та мікросервісів.
- Реального часу: реалізація чатів, онлайн-ігор та інших додатків через SignalR.
- Хмарні додатки: додатки з високим масштабуванням і доступністю.

Особливості ASP.NET Core [38]:

1. Кросплатформність:

ASP.NET Core дозволяє створювати додатки, які можуть працювати на Windows, Linux та macOS. Це досягається завдяки використанню .NET Runtime, що має однакову реалізацію для всіх платформ.

- Розробники можуть використовувати будь-яке середовище розробки (наприклад, Visual Studio, Visual Studio Code або Rider).
- Завдяки кросплатформності можна розгортати додатки на різних платформах, включаючи хмарні сервіси та контейнерні середовища (Docker).

2. Висока продуктивність:

ASP.NET Core – один із найшвидших фреймворків для веб-додатків, що підтверджується незалежними тестами (наприклад, TechEmpower Benchmarks).

- Kestrel: вбудований високопродуктивний веб-сервер, оптимізований для роботи з великим навантаженням.
- Використання асинхронної моделі обробки запитів через async/await, що знижує витрати на потоки та ресурси.
- Оптимізація роботи з пам'яттю завдяки механізмам Garbage Collection і структурі .NET Core.

3. Модульна архітектура:

ASP.NET Core дозволяє використовувати тільки ті компоненти, які потрібні для конкретного проєкту.

- Middleware**:** система обробки запитів і відповідей побудована на послідовності проміжних компонентів, які можна додавати або видаляти за необхідності.
- Завантаження залежностей виконується через NuGet, що дозволяє додавати тільки необхідні бібліотеки.

4. Open Source:

ASP.NET Core має відкритий вихідний код, доступний на GitHub. Це забезпечує:

- Швидке усунення помилок завдяки внескам спільноти.
- Прозорість у розробці фреймворку.
- Можливість кастомізації: якщо потрібно, можна змінити код фреймворку для своїх потреб.

5. Хмарна інтеграція:

ASP.NET Core тісно інтегрується з хмарними сервісами Microsoft Azure:

- Підтримка Azure App Services для швидкого розгортання додатків.
- Використання Azure Functions для безсерверних обчислень.
- Інтеграція з Azure Key Vault для зберігання конфіденційних даних, таких як паролі, сертифікати тощо.

6. Розширювана модель:

Фреймворк побудований із врахуванням принципу відкритості до розширення:

- Легко додаються нові middleware-компоненти, які можна налаштовувати залежно від потреб.
- ASP.NET Core має вбудовану підтримку Dependency Injection (DI), яка спрощує управління залежностями та модульне тестування.

7. Вбудована безпека:

Безпека вбудована на всіх рівнях:

- ASP.NET Core Identity — готовий механізм для автентифікації та управління користувачами.
- Інтеграція з OAuth та OpenID Connect для реалізації SSO (єдиного входу).
- Вбудовані механізми захисту від поширених атак:
  - Захист від XSS (міжсайтовий скриптинг) через автоматичну екранування даних у Razor Views.
  - CSRF-захист (підробка міжсайтових запитів) через перевірку токенів.
  - Захист від SQL-ін'єкцій завдяки використанню ORM (наприклад, Entity Framework Core).

8. Сучасні веб-технології:

ASP.NET Core підтримує широкий спектр технологій для створення інтерактивних веб-додатків:

- Інтеграція з популярними SPA-фреймворками (React, Angular, Vue).
- Blazor – технологія для створення інтерфейсів на основі C#, що працюють у браузері через WebAssembly.
- SignalR – бібліотека для реалізації функціоналу реального часу (наприклад, чати, нотифікації).

9. Тестування та налагодження:

ASP.NET Core надає потужні інструменти для тестування та налагодження:

- Модульні тести: через бібліотеки MSTest, xUnit або NUnit.
- Інтеграційні тести: для перевірки взаємодії між компонентами.

- Вбудовані можливості логування (через Serilog, NLog, або інші провайдери).

Можливості для створення масштабованих додатків:

ASP.NET Core забезпечує широкі можливості для створення масштабованих додатків завдяки своїй сучасній архітектурі, гнучкості та інтеграції з потужними інструментами.

Однією з ключових особливостей є масштабування. Горизонтальне масштабування дозволяє додавати нові інстанси додатка через інтеграцію з Docker і Kubernetes, що забезпечує безперебійну роботу навіть при значному зростанні навантаження. Вертикальне масштабування дає можливість збільшувати ресурси сервера для підвищення продуктивності.

Мікросервісна архітектура сприяє розподіленості додатків, де кожен мікросервіс може працювати незалежно, бути масштабованим і розгортатися окремо. Це забезпечує стабільність системи і швидку адаптацію до змін [23].

Інтеграція з веб-серверами (Kestrel, IIS, Nginx, Apache) робить ASP.NET Core універсальним рішенням для розгортання додатків у різних середовищах.

Підтримка роботи з базами даних реалізується через Entity Framework Core, який дозволяє взаємодіяти як із SQL, так і з NoSQL базами. Наприклад, Azure SQL чи Cosmos DB можуть використовуватися для автоматичного масштабування з урахуванням поточних потреб.

Щоб забезпечити високу продуктивність, ASP.NET Core має ефективні механізми кешування, такі як Redis або SQL Server, що дозволяють знижувати навантаження на бази даних і прискорювати обробку запитів.

Система моніторингу та логування забезпечується за допомогою інструментів на зразок Azure Application Insights, Serilog або ELK Stack. Вони дозволяють детально аналізувати продуктивність системи та оперативно реагувати на помилки чи збої [37].

Завдяки асинхронній обробці запитів, додаток може обробляти тисячі одночасних запитів, що є критично важливим для високонавантажених систем.

Ще однією перевагою є API-first підхід, який спрощує розробку RESTful API з автоматичним генеруванням документації через Swagger/OpenAPI. Це дозволяє легко

інтегрувати серверну частину з клієнтськими додатками, такими як мобільні застосунки чи веб-інтерфейси.

На окрему увагу заслуговують інструменти безпеки. Наприклад, Azure Active Directory дозволяє захистити доступ до ресурсів, а Key Vault – надійно зберігати секрети та ключі.

Для ілюстрації цих можливостей доцільно створити схему, що відображає взаємодію компонентів: мікросервісів, баз даних, кешування, серверів та інструментів моніторингу. Така схема допоможе наочно зрозуміти переваги масштабованості та гнучкості ASP.NET Core.

1.1.2 Архітектура ASP.NET Core та моделі взаємодії з базою даних

ASP.NET Core підтримує модульну, гнучку архітектуру, яка дозволяє розробникам організовувати код для забезпечення масштабованості, тестованості та розширюваності.

Чиста архітектура (Clean Architecture) – це підхід до проєктування програмного забезпечення, який забезпечує гнучкість, масштабованість і легкість підтримки коду. Цей підхід був популяризований Робертом Мартіном (Robert C. Martin), також відомим як "дядько Боб". Основною метою чистої архітектури є відділення бізнес-логіки від деталей реалізації, таких як база даних, веб-фреймворки чи зовнішні сервіси [32].

Можна виділити такі основні шари:

1. Шар домену (Domain Layer)

Шар домену в Чистій архітектурі є однією з найважливіших складових, оскільки він відповідає за реалізацію основної бізнес-логіки додатку. Його основне завдання – забезпечити повну ізоляцію від деталей зовнішніх систем і технологій, що дає змогу зберігати бізнес-правила чистими та незалежними від інших частин програми [33].

У шарі домену зберігаються моделі, які відображають бізнес-об'єкти системи. Це можуть бути сутності (Entities), які мають унікальні ідентифікатори і змінюють свій стан, і об'єкти значення (Value Objects), які є невід'ємною частиною моделей і

змінюються разом із ними. Наприклад, у системі для управління замовленнями сутностями можуть бути Order (замовлення) або Customer (клієнт), а об'єктами значення – Order Item (позиція в замовленні), який не має унікального ідентифікатора і змінюється разом із замовленням.

Ще однією важливою частиною шару домену є інтерфейси для доступу до бізнес-логіки, наприклад, сервіси та репозиторії. Сервіси містять бізнес-операції, які реалізують основну логіку додатку, наприклад, створення замовлення, обчислення вартості або обробка платежу. Репозиторії ж надають доступ до даних, зберігаючи та отримуючи сутності з бази даних або іншої системи збереження інформації.

Шар домену має важливу особливість – він не має залежностей від зовнішніх систем або технологій. Це означає, що бізнес-логіка, яка міститься в ньому, може працювати незалежно від конкретних рішень щодо баз даних, веб-серверів або фреймворків. Інші шари, такі як шар додатків чи шар інфраструктури, звертаються до шару домену, але не навпаки. Такий підхід дозволяє зберігати логіку чистою і легко адаптованою до змін технологій чи вимог.

Ключовою перевагою шару домену є його гнучкість і здатність до адаптації. Оскільки він не залежить від технологій, його можна легко масштабувати або змінювати без впливу на інші частини програми. Це дозволяє зберігати програму стабільною і готовою до змін, навіть коли зовнішні компоненти змінюються або оновлюються [32].

Загалом, шар домену забезпечує чітке і структуроване управління бізнес-логікою додатку, дозволяючи досягати високої підтримуваності та масштабованості системи. Це також полегшує тестування та підтримку додатку, оскільки бізнес-правила ізольовані від деталей реалізації технологій, таких як бази даних чи інтерфейси користувача.

2. Шар додатку (Application Layer)

Шар додатку служить як міст між доменною логікою та зовнішніми системами. Завдяки цьому він дозволяє організувати ефективну взаємодію з різними клієнтами (веб-інтерфейсами, мобільними додатками, сторонніми API тощо), зберігаючи при цьому чітке розмежування між бізнес-логікою та зовнішніми деталями реалізації.

Оскільки цей шар не містить бізнес-логіки, він зберігає гнучкість системи, спрощуючи підтримку та масштабування [32].

Таким чином, шар додатка є важливим елементом, що гарантує організацію взаємодії з зовнішніми запитами та правильно спрямовує їх до відповідних бізнес-операцій, визначених у шарі домену.

Основні елементи шару додатків:

- Сервіси додатка**:** це класи, що виконують бізнес-процеси, використовуючи об’єкти та методи шару домену. Вони часто реалізують конкретні сценарії чи робочі потоки. Наприклад, сервіс для управління користувачами може мати методи для реєстрації користувача, автентифікації, оновлення профілю тощо.
- Команди та запити**:** шар додатка також може містити обробку запитів через Command та Query патерни (CQRS – Command Query Responsibility Segregation). Запити (Query) використовуються для отримання даних, а команди (Command) для зміни стану системи. Це дозволяє чітко розмежувати логіку для читання та запису даних.
- Інтерфейси для взаємодії з інфраструктурними компонентами**:** шар додатка не містить безпосередньо реалізацій доступу до бази даних чи зовнішніх сервісів. Він визначає інтерфейси для взаємодії з інфраструктурними компонентами, які реалізуються у шарі інфраструктури (наприклад, репозиторії для доступу до бази даних або сервіси для роботи з файлами).
- Трансфер об’єктів (DTO)**:** для передачі даних між шарами, наприклад, між контролерами та сервісами додатка, часто використовуються об’єкти передачі даних (DTO). Вони дозволяють інкапсулювати дані, які потрібно передати, зберігаючи внутрішню логіку роботи системи незалежною від зовнішніх запитів.

Особливості шару додатка:

- Ізоляція бізнес-логіки від деталей інфраструктури: шар додатка містить логіку, яка не залежить від конкретних технологій чи реалізацій. Це означає, що зміна технологій або інфраструктури не повинна впливати на бізнес-правила та робочі процеси, визначені в бізнес-логіці.

- Виконання сценаріїв використання: Шар додатка організовує і координує виконання певних сценаріїв або задач, використовуючи об’єкти і сервіси з шару домену. Це можуть бути, наприклад, сценарії покупки, обробки замовлень, управління користувачами або навіть складніші бізнес-процеси, що потребують кількох кроків виконання.
- Використання шаблонів проєктування: Для реалізації шар додатка часто використовує популярні шаблони проєктування, такі як Factory, Strategy, Observer та інші. Це дозволяє створювати гнучкі, підтримувані та масштабовані рішення для виконання бізнес-процесів.

3. Шар інфраструктури (Infrastructure Layer)

Шар інфраструктури (Infrastructure Layer) відповідає за реалізацію технічних аспектів, які забезпечують роботу програми, таких як доступ до баз даних, взаємодія з зовнішніми API, файлові операції, логування та кешування. Він реалізує інтерфейси, визначені в доменному та прикладному шарах, забезпечуючи їхні конкретні реалізації [32].

У цьому шарі можуть бути:

- Репозиторії для доступу до баз даних (через Entity Framework або інші технології).
- Реалізації сервісів для роботи з файлами, поштовими сервісами, повідомленнями.
- Інтеграція з зовнішніми API та сторонніми сервісами.
- Логування та моніторинг (наприклад, Serilog або NLog).
- Кешування (наприклад, Redis).

Шар інфраструктури ізолює додаток від деталей реалізації технологій, що дозволяє легко змінювати технологічні рішення без впливу на бізнес-логіку.

4. Шар презентації (Presentation Layer)

Шар презентації (Presentation Layer) відповідає за взаємодію з користувачем або клієнтськими додатками. Цей шар має на меті забезпечити зручний і ефективний спосіб представлення даних користувачеві та отримання вводу від нього. Всі інтерфейси, через які користувач взаємодіє з додатком, належать до цього шару [33].

Основні функції цього шару:

- Взаємодія з користувачем через веб-інтерфейс (наприклад, за допомогою MVC або Razor Pages у ASP.NET Core) або через мобільний інтерфейс.
- Обробка запитів від користувача та відправка їх до прикладного шару для виконання бізнес-логіки.
- Відображення результатів, отриманих від бізнес-логіки (наприклад, через HTML, JSON або інші формати).
- Валідація даних, що надходять від користувача перед їх передачею до прикладного шару.

Шар презентації часто використовує шаблони проєктування, такі як MVC (Model-View-Controller) або MVVM (Model-View-ViewModel), для організації коду і забезпечення чіткої роздільності відповідальностей.

1.1.3 Entity Framework для роботи з базами даних у ASP.NET Core

Entity Framework (EF) є потужним ORM (Object-Relational Mapping) інструментом, який дозволяє розробникам працювати з базами даних, використовуючи об'єкти в коді, що спрощує роботу з даними та забезпечує автоматичну синхронізацію між об'єктами та таблицями в базі даних. У контексті ASP.NET Core, EF Core є основним інструментом для роботи з базами даних, і він надає безліч можливостей для інтеграції, масштабування та оптимізації додатків [28].

Основні принципи та можливості використання Entity Framework в ASP.NET Core:

- Моделювання даних через C# класи: EF Core дозволяє створювати класи, які відображають структуру таблиць бази даних. Це означає, що для кожної сутності в додатку (наприклад, User, Product, Order) створюється відповідний клас. Ці класи можна використовувати для збереження, оновлення, видалення та вибірки даних.
- Міграції бази даних: EF Core підтримує міграції, що дозволяють автоматично створювати та оновлювати схему бази даних відповідно до змін в коді моделей. Міграції забезпечують контроль версій бази даних, полегшуючи оновлення

схеми без втрати даних. Для цього використовуються команди, такі як Add-Migration та Update-Database.

- LINQ-запити: EF Core підтримує LINQ (Language Integrated Query), що дозволяє писати запити до бази даних за допомогою C#. Це забезпечує високу зручність при створенні складних запитів та фільтрації даних без необхідності писати SQL-запити вручну.
- Зв'язки між сутностями: EF Core підтримує всі основні типи зв'язків між таблицями, такі як:
- Один до одного (1:1),

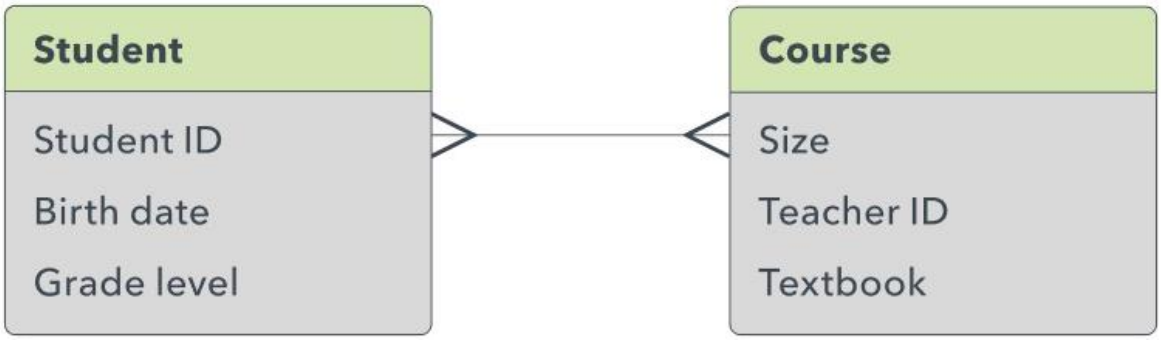


Рис. 1.1. Зв’язок один до одного

- Один до багатьох (1: N),

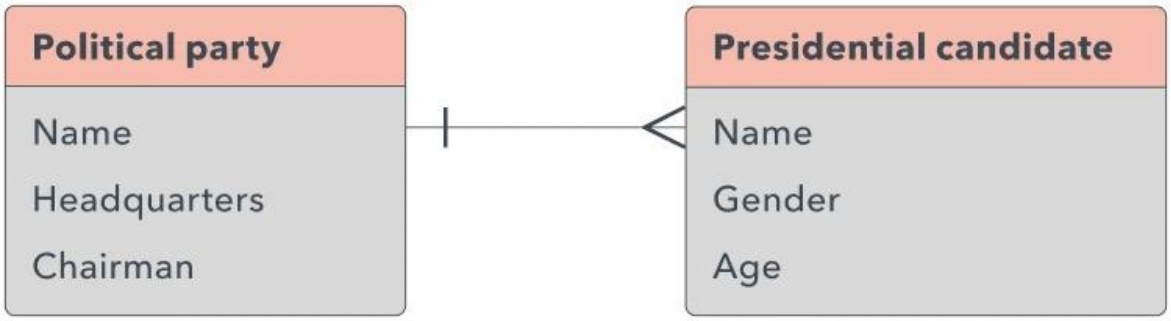


Рис. 1.2. Зв’язок один до багатьох

- Багато до багатьох (N: N). Ці зв'язки можна налаштовувати через Fluent API або атрибути, що дозволяє точно визначити, як повинні бути пов'язані таблиці та сутності.

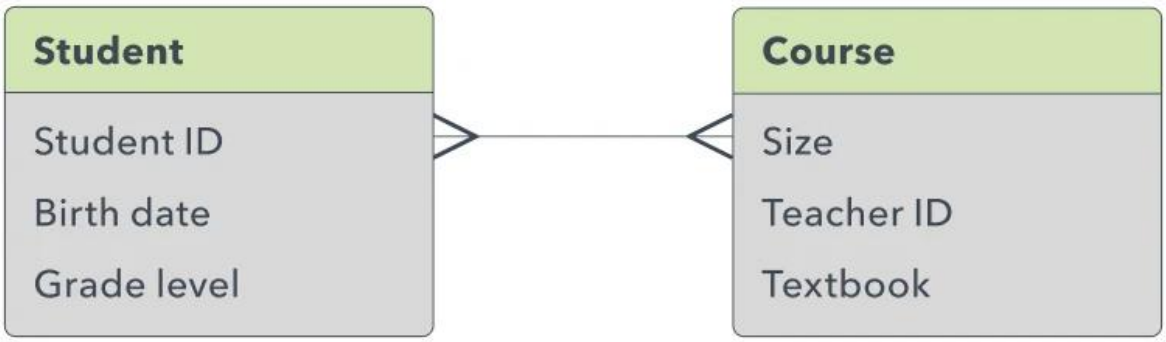


Рис. 1.3. Зв’язок один до багатьох

- Асинхронні операції: EF Core підтримує асинхронні операції, що дозволяє ефективно обробляти запити до бази даних без блокування потоку виконання. Це особливо корисно для веб-додатків, де час відповіді має критичне значення, і високий рівень одночасних запитів.
- Підтримка кількох баз даних: EF Core підтримує роботу з різними типами баз даних, такими як SQL Server, SQLite, PostgreSQL, MySQL, а також NoSQL бази даних, як-от Cosmos DB. Це забезпечує гнучкість у виборі бази даних для проєкту.
- Трасування та відладка: EF Core дозволяє відстежувати SQL-запити, що генеруються на основі LINQ-запитів, що допомагає відладити запити та оптимізувати їх продуктивність.

1.2 Angular і TypeScript та їх зв’язок з backend частиною

1.2.1 Основи Angular

Angular – це потужний і гнучкий фреймворк для розробки односторінкових веб-додатків (SPA) на платформі TypeScript, який активно використовується для створення сучасних, високопродуктивних веб-додатків [24].

Angular був розроблений і підтримується компанією Google та має велику екосистему інструментів і бібліотек, що дозволяють вирішувати різні завдання розробки, включаючи маршрутизацію, управління станом, роботу з формами, асинхронною обробкою та багато іншого.

Однією з головних особливостей Angular є його компонентно-орієнтована архітектура, що дає змогу створювати додатки, які є масштабованими та легко підтримуваними. Кожен компонент є автономною одиницею інтерфейсу, яка містить як шаблон (HTML), так і стилі (CSS), а також логіку (TypeScript) [24].

Основні особливості Angular:

1. Модульність (Modules): angular побудований на основі модулів, кожен з яких є контейнером для компонентів, сервісів та інших елементів додатка. Всі елементи додатка можна організувати в окремі модулі, що дозволяє значно покращити підтримуваність і масштабованість. Основним модулем в Angular є root module (наприклад, AppModule), який є точкою входу додатка.

2. Компоненти (Components): компоненти – це основні будівельні блоки в Angular. Кожен компонент відповідає за одну частину інтерфейсу користувача та включає три основні частини:

- HTML-шаблон: визначає структуру елементів на сторінці.
- CSS: визначає стилі для компонента.
- TypeScript: містить логіку компонента, яка управляє даними і обробляє події.

3. Директиви (Directives): Директиви в Angular є спеціальними інструкціями, які змінюють поведінку DOM-елементів. Вони поділяються на кілька типів:

- Structural directives змінюють структуру DOM (наприклад, ngIf, ngFor).
- Attribute directives змінюють зовнішній вигляд або поведінку елементів (наприклад, ngClass, ngStyle).

4. Сервіси та Ін'єкція Залежностей (Services and Dependency Injection): в Angular сервіси використовуються для централізованої обробки даних, логіки бізнесу або запитів до серверів. Вони часто використовуються разом з ін'єкцією залежностей (DI), що дозволяє Angular автоматично впроваджувати залежності в компоненти чи інші сервіси. Це зменшує кількість ручного коду та підвищує тестованість.

5. Роутинг (Routing): angular має вбудовану систему маршрутизації, яка дозволяє створювати динамічні сторінки та переходити між ними без

перезавантаження всієї сторінки. Це забезпечує створення односторінкових веб-додатків. Кожен маршрут відповідає за відображення певного компонента в залежності від URL.

6. Форми (Forms): angular підтримує два основні підходи для роботи з формами:

- Template-driven forms: підходить для простих форм, де більшість логіки визначається безпосередньо в шаблоні.
- Reactive forms: більш потужний підхід для складніших форм, де вся логіка і стан форми управляються через TypeScript.

7. RxJS та реактивне програмування: Angular активно використовує RxJS – бібліотеку для обробки асинхронних подій через потоки даних. Це дозволяє працювати з асинхронними операціями, такими як запити до серверів або обробка подій, у більш декларативний спосіб. RxJS використовується для обробки промісів, HTTP-запитів, таймерів тощо.

8. Тестування: Angular включає потужні інструменти для тестування додатків. Використовуючи Jasmine для написання тестів і Karma для запуску тестів в браузері, Angular дозволяє розробникам легко перевіряти компоненти, сервіси та інші частини додатка на правильність виконання.

1.2.2 Переваги використання TypeScript у розробці Angular додатків

Використання TypeScript у розробці Angular додатків надає низку значних переваг, які сприяють підвищенню ефективності, зручності розробки та підтримуваності коду. TypeScript є надбудовою над JavaScript, що додає статичну типізацію, покращує обробку помилок на етапі компіляції та сприяє організованому підходу до розробки, особливо в контексті великих і складних проєктів, таких як Angular додатки [13].

Однією з основних переваг TypeScript є статична типізація. Вона дозволяє визначати типи змінних, функцій та класів під час написання коду, що дає змогу виявляти помилки ще на етапі компіляції, а не під час виконання програми. Це значно знижує ймовірність виникнення багів, пов'язаних із неправильними типами даних,

таких як спроби виконати математичні операції з рядками або доступ до невизначених властивостей об'єктів. В Angular додатках, де часто працюють з різними даними (від запитів до серверів до обробки користувацьких введень), статична типізація забезпечує надійність і точність коду, що особливо важливо при роботі з великою кількістю компонентів та сервісів.

Іншою важливою перевагою є підтримка класів, інтерфейсів та абстрактних класів. TypeScript надає можливість використовувати сучасні об'єктно-орієнтовані концепції в JavaScript, такі як наслідування, інкапсуляція та поліморфізм, що дає змогу створювати більш організований і структурований код. Для Angular це важливо, оскільки фреймворк використовує компонентно-орієнтовану архітектуру, де кожен компонент є класом. Використання TypeScript дозволяє чітко визначити структуру компонентів, їх залежності та взаємодії через інтерфейси, що значно покращує підтримуваність і розширюваність проєктів.

TypeScript також надає можливість використовувати декоратори, що є важливим аспектом в Angular. Декоратори дозволяють додавати метадані до класів, властивостей, методів або параметрів функцій, що дозволяє зручно визначати різні аспекти поведінки Angular компонентів, сервісів або директив. Наприклад, декоратор *@Component* використовується для визначення компонентів, а декоратор *@Injectable* – для вказівки на сервіси, які можна впроваджувати через ін'єкцію залежностей. Це забезпечує зручний спосіб організації та структурування коду, що спрощує розуміння і підтримку великих додатків [14].

Однією з найбільш цінних функцій TypeScript є підтримка IntelliSense та автодоповнення в редакторах коду, таких як Visual Studio Code. Оскільки TypeScript дозволяє визначити типи даних, функцій та методів, редактори можуть надавати точну підказку при написанні коду, що значно пришвидшує процес розробки. Розробники отримують підказки не тільки щодо імен змінних і функцій, а й щодо їх типів, що допомагає уникати багів і не витрачати час на помилки. Також TypeScript забезпечує зручне і зрозуміле автодоповнення для складних об'єктів і структур, що дозволяє працювати з великою кількістю компонентів і модулів без страху забути важливі деталі.

TypeScript покращує тестованість коду завдяки своїй типізації і строгим правилам. Задача тестування в Angular додатках ускладнюється через наявність численних залежностей між компонентами та сервісами. TypeScript дозволяє чітко визначати контракти між компонентами, що спрощує створення юніт-тестів і допомагає тестувальникам зрозуміти, які саме дані і в якому вигляді повинні бути передані між компонентами. Це дозволяє швидше писати тести і знижує ймовірність виникнення помилок у майбутньому.

Також TypeScript підтримує нові можливості ECMAScript, такі як async/await для роботи з асинхронними операціями [14]. Це забезпечує легку інтеграцію з сучасними методами роботи з асинхронним кодом, що особливо важливо для Angular додатків, де часто виникають ситуації, пов'язані з асинхронними запитами до серверів, обробкою форм або зовнішніми бібліотеками.

Типізація також спрощує інтеграцію з іншими фреймворками та бібліотеками, оскільки дозволяє чітко визначити, як і які типи даних мають бути використані в межах взаємодії. Більшість популярних бібліотек, зокрема ті, що використовуються в Angular, надають типи для TypeScript, що дозволяє безпечно використовувати сторонні компоненти без ризику неочікуваних помилок.

Загалом, використання TypeScript в розробці Angular додатків значно полегшує процес розробки, підвищує надійність та знижує ймовірність помилок. Це потужний інструмент, який дозволяє розробникам створювати чистий, ефективний і підтримуваний код, що є важливим аспектом для масштабованих та складних веб-додатків.

1.2.3 Зв’язок між фронтендом на Angular та бекендом на ASP.NET Core

Зв'язок між фронтендом на Angular і бекендом на ASP.NET Core є важливою частиною архітектури сучасних веб-додатків. Angular і ASP.NET Core взаємодіють через HTTP-запити, де Angular виступає клієнтським додатком, а ASP.NET Core – сервером, який обробляє ці запити. Для цього використовуються різні механізми, зокрема RESTful API та JSON для передачі даних.

Взаємодія через RESTful API

Основним способом комунікації між фронтендом на Angular та бекендом на ASP.NET Core є RESTful API. В Angular фронтенд надсилає HTTP-запити (GET, POST, PUT, DELETE тощо) до серверної частини, яка реалізована на ASP.NET Core. API визначає набір маршрутів для доступу до різних ресурсів (наприклад, користувачів, продуктів, замовлень) на сервері, і ці маршрути повинні бути спроєктовані таким чином, щоб забезпечити правильну обробку запитів і відповідей.

Відправка запитів з Angular

Angular використовує HTTP-клієнт для відправки запитів на сервер. В Angular є спеціальний сервіс HttpClient, який дозволяє робити запити до бекенду. Це можуть бути прості запити на отримання даних або більш складні запити, що включають передачу даних для створення або оновлення ресурсів.

Наприклад, для отримання списку користувачів з бекенду на ASP.NET Core, в Angular можна використати наступний код:

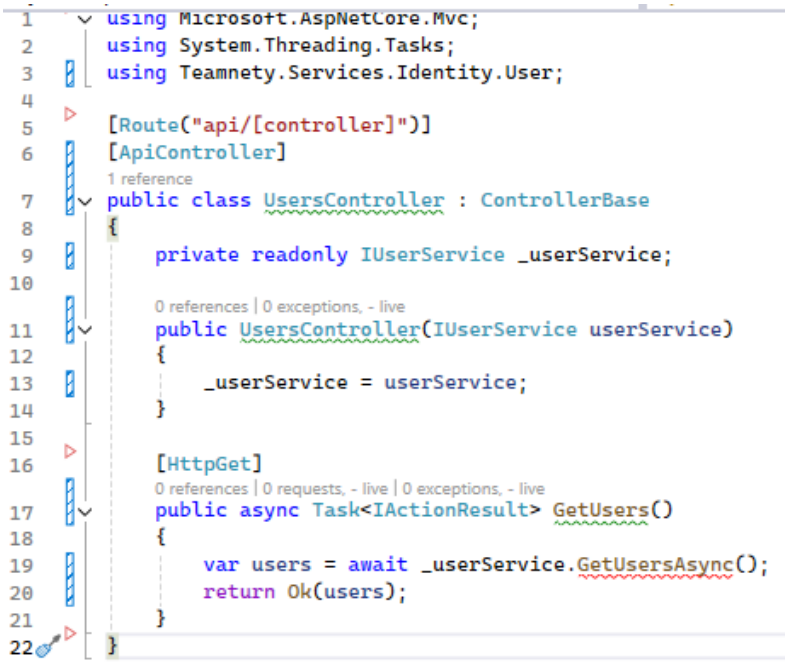
```
1   import { HttpClient } from '@angular/common/http';
2   import { Injectable } from '@angular/core';
3
4   @Injectable({
5     providedIn: 'root'
6   })
7   export class UserService {
8     constructor(private http: HttpClient) {}
9
10    getUsers() {
11      return this.http.get('https://example.com/api/users');
12    }
13  }
```

Рис. 1.4. Опис сервісу User Service

Обробка запитів на бекенді (ASP.NET Core)

На стороні бекенду в ASP.NET Core для обробки запитів і визначення маршрутів використовуються контролери. Контролери — це класи, що містять методи, які відповідають на HTTP-запити з певними маршрутами. Для обробки

запиту на отримання списку користувачів з бекенду можна створити наступний контролер:



```
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;
using Teamnety.Services.Identity.User;

[Route("api/[controller]")]
[ApiController]
1 reference
public class UsersController : ControllerBase
{
    private readonly IUserService _userService;

    0 references | 0 exceptions, - live
    public UsersController(IUserService userService)
    {
        _userService = userService;
    }

    [HttpGet]
    0 references | 0 requests, - live | 0 exceptions, - live
    public async Task<IActionResult> GetUsers()
    {
        var users = await _userService.GetUsersAsync();
        return Ok(users);
    }
}
```

Рис. 1.5. Опис контролеру для користувачів

Формат даних (JSON)

Для взаємодії між клієнтською та серверною частинами зазвичай використовується формат JSON, оскільки він є легким для читання та парсингу як на стороні клієнта, так і на сервері. В ASP.NET Core за замовчуванням відповіді серіалізуються в формат JSON, якщо це не налаштовано інакше.

У відповідь на запит від Angular сервер на ASP.NET Core може повернути JSON-об'єкт, який потім буде оброблено на фронтенді:

```
{
    "id": 1,
    "name": "John Doe",
    "email": "john@example.com"
},
{
    "id": 2,
    "name": "Jane Smith",
    "email": "jane@example.com"
}
```

Рис. 1.6. Формат даних за допомогою JSON

Захист і аутентифікація

У реальних додатках часто потрібно захищати доступ до ресурсів. Одним із найбільш популярних способів захисту є використання JWT (JSON Web Tokens) для аутентифікації та авторизації. На стороні бекенду на ASP.NET Core можна налаштувати аутентифікацію через JWT, а на фронтенді – передавати токен з кожним запитом у заголовках. Це дозволяє серверу перевіряти, чи є користувач авторизованим перед тим, як надавати доступ до захищених ресурсів.

```
addJob(jobModel: any): Observable<JobPublicModel> {
    return this._httpClientService.post(HttpClientService.ADD_JOB, jobModel);
}
```

Рис. 1.7. Приклад надсилання запиту зі сторони клієнта

Взаємодія через WebSocket

Для реального часу, коли потрібна двостороння комунікація між клієнтом і сервером (наприклад, чати або повідомлення в реальному часі), можна використовувати WebSocket. В Angular можна інтегрувати бібліотеки для роботи з WebSocket, а на стороні сервера в ASP.NET Core – налаштувати SignalR, який є потужним інструментом для реалізації таких функцій.

1.3 Розробка мобільних застосунків для iOS і Android

1.3.1 Підходи до розробки мобільних додатків на різних платформах

У сучасному цифровому світі мобільні додатки є важливим інструментом для взаємодії з користувачами. Додаток для пошуку роботи має забезпечувати зручність, доступність і функціональність для кандидатів, що шукають вакансії. Розробка такого додатка для платформ iOS та Android може бути реалізована за різними підходами залежно від вимог проєкту та аудиторії.

Нативна розробка для iOS та Android

Нативна розробка передбачає створення окремих додатків для кожної платформи, використовуючи Swift для iOS та Kotlin для Android. Це дозволяє

реалізувати максимально оптимізований і продуктивний функціонал. Такий підхід забезпечує доступ до повного спектру можливостей пристрою, зокрема пуш-сповіщень, геолокації та інтеграції з іншими сервісами.

Для пошуку роботи це означає швидкий пошук вакансій, персоналізовані рекомендації, нагадування про дедлайни подачі та інтеграцію з календарем користувача. Однак, нативна розробка вимагає більших затрат часу та ресурсів, оскільки потребує окремих команд для розробки кожної версії додатка.

Кросплатформна розробка

Кросплатформні технології, такі як Flutter або React Native, дозволяють створювати один код для обох платформ. Це значно скорочує час розробки й зменшує бюджет. У контексті додатка для пошуку роботи такий підхід може бути особливо корисним, якщо проєкт обмежений у фінансуванні, але має забезпечувати якісне користувацьке середовище.

Кросплатформні рішення дозволяють створювати сучасний інтерфейс, який буде однаково виглядати на обох платформах. Наприклад, функція перегляду вакансій або завантаження резюме може бути реалізована через загальні компоненти, що забезпечить однакову функціональність для користувачів iOS та Android.

Прогресивні вебдодатки (PWA)

Прогресивні вебдодатки – це ще один підхід, що дозволяє створювати додаток, доступний через браузер на будь-якому пристрої. Для пошуку роботи PWA може стати економічно вигідним рішенням. Це забезпечить користувачам доступ до вакансій та їхнього профілю без необхідності завантаження додатка з магазину.

Хоча PWA має обмежений доступ до можливостей пристрою, це може бути ефективним варіантом для тестування ринку чи швидкого запуску MVP.

У нашому випадку реалізація проєкту виконана за допомогою нативної розробки для платформ iOS та Android. Цей підхід забезпечує максимальну продуктивність додатків і дозволяє використовувати всі унікальні можливості кожної платформи.

Для розробки додатка на iOS була використана мова програмування Swift, що забезпечує високу швидкість роботи, зручну інтеграцію з екосистемою Apple

(наприклад, інтеграція з календарем чи віджетами), а також можливість використання сучасних функцій iOS, таких як Face ID або Touch ID для авторизації користувачів.

Додаток для Android був створений за допомогою Kotlin, що дозволяє ефективно працювати з пристроями різних виробників і версій операційної системи. Завдяки нативному підходу вдалося забезпечити оптимальну продуктивність навіть на пристроях середнього рівня.

Особливу увагу приділено користувацькому досвіду (UX/UI), що було реалізовано відповідно до рекомендацій Apple Human Interface Guidelines і Google Material Design. Це дозволило створити інтерфейс, що є інтуїтивно зрозумілим для користувачів кожної з платформ.

Нативна розробка також забезпечила безперебійну роботу таких ключових функцій:

- Пуш-сповіщення для швидкого інформування користувачів про нові вакансії, зміни у статусі їхніх заявок чи персоналізовані рекомендації.
- Геолокація, яка дозволяє шукати вакансії поруч із місцезнаходженням користувача.
- Офлайн-режим, що дає змогу переглядати раніше завантажені вакансії або створювати чернетки резюме навіть без доступу до інтернету.

Нативний підхід також полегшує інтеграцію з бекендом на ASP.NET Core, забезпечуючи швидке та надійне обмінювання даними через RESTful API. Це дозволяє реалізовувати функції, пов’язані з пошуком роботи, завантаженням резюме, рекомендаціями вакансій та комунікацією з роботодавцями, з максимальною ефективністю.

У результаті, нативна розробка стала ідеальним вибором для проєкту, спрямованого на створення високоякісного додатка, здатного задовольнити потреби сучасного ринку праці.

1.3.2 Azure для хмарної інтеграції мобільних додатків

Використання Azure для хмарної інтеграції мобільних додатків стало невід'ємною частиною сучасної розробки, особливо у випадках, коли потрібна

надійна, масштабована та безпечна платформа для зберігання даних і обробки запитів. У контексті нашого проєкту, спрямованого на пошук роботи для кандидатів, Azure забезпечує ключову інфраструктуру, яка підтримує роботу мобільних додатків на платформах iOS та Android [28].

Однією з основних функцій Azure є її здатність до безшовної інтеграції з мобільними додатками завдяки таким сервісам:

1. Azure App Service дозволяє розробляти та розгортати бекенд, який обробляє запити від мобільних клієнтів. Цей сервіс забезпечує швидке масштабування відповідно до навантаження, а також підтримує RESTful API, який використовується для передачі даних між мобільними додатками та сервером.

2. Azure SQL Database надає надійний механізм для зберігання даних, таких як профілі кандидатів, інформація про вакансії та історія пошуку. Завдяки вбудованим можливостям масштабування ця база даних може ефективно обслуговувати зростаючу кількість користувачів.

3. Azure Blob Storage використовується для зберігання великих обсягів даних, таких як резюме користувачів у форматі PDF або мультимедійні файли. Це дозволяє організувати надійне зберігання та швидкий доступ до файлів через мобільні додатки.

4. Push Notifications через Azure Notification Hubs дозволяють оперативно надсилати повідомлення користувачам мобільних додатків. Наприклад, вони отримують сповіщення про нові вакансії, зміну статусу заявки або персоналізовані рекомендації.

5. Azure Active Directory B2C забезпечує управління користувацькими обліковими записами та автентифікацією, включаючи соціальний вхід через Facebook, Google або Microsoft. Ця функція не лише підвищує безпеку, але й спрощує процес реєстрації та входу для користувачів.

6. Azure Monitor та Application Insights використовуються для моніторингу продуктивності додатків. Це дозволяє відстежувати поведінку користувачів, виявляти проблеми у роботі системи та оптимізувати її.

Однією з ключових переваг Azure є її глобальна інфраструктура, яка забезпечує низькі затримки для користувачів, незалежно від їхнього місця розташування. Наприклад, користувачі, що знаходяться в різних частинах світу, можуть отримувати доступ до тих самих сервісів із мінімальною затримкою завдяки розташуванню серверів Azure у різних регіонах.

Для мобільних додатків, орієнтованих на пошук роботи, інтеграція з Azure дозволяє забезпечити стабільну роботу, навіть при значному збільшенні кількості користувачів. Це особливо важливо у пікові періоди, коли зростає активність як кандидатів, так і роботодавців.

Таким чином, використання Azure для хмарної інтеграції є стратегічно важливим рішенням, яке дозволяє створити надійний та масштабований сервіс для мобільних додатків, сприяючи успішному досягненню цілей проєкту.

Висновки до розділу

У розділі проведено огляд сучасних технологій розробки веб- і мобільних застосунків, які сьогодні є основою для створення високопродуктивних і зручних програмних рішень. Проаналізовано популярні фреймворки, бібліотеки та мови програмування, зокрема Angular, React, Flutter, які забезпечують ефективність розробки, масштабованість, адаптивність інтерфейсів та кросплатформеність.

Особливу увагу приділено використанню хмарних сервісів, таких як Microsoft Azure та AWS, які забезпечують зберігання даних, автоматичне масштабування та високу доступність. Також розглянуто підходи до розробки API для інтеграції між клієнтською і серверною частинами, використання технологій REST та GraphQL, які сприяють зручній та ефективній взаємодії компонентів.

На основі аналізу визначено, що застосування сучасних технологій і підходів дозволяє створювати адаптивні, швидкі й безпечні застосунки, що відповідають вимогам користувачів і бізнесу. Результати огляду стали фундаментом для обґрунтування вибору технологій у подальшій практичній реалізації проєкту.

РОЗДІЛ 2 АНАЛІЗ ХМАРНИХ ТЕХНОЛОГІЙ ТА МЕТОДИ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ДАНИХ

2.1 Хмарні технології та їх застосування в розробці застосунків

2.1.1 Переваги використання хмарних технологій

Хмарні технології – це модель обчислень, яка передбачає надання послуг (інфраструктури, платформи, програмного забезпечення) через Інтернет. У хмарному середовищі ресурси, такі як сервери, сховища даних, мережі та програми, доступні користувачам на вимогу, без необхідності володіння фізичним обладнанням чи програмним забезпеченням. Основною ідеєю хмарних обчислень є розподіл ресурсів, що дозволяє забезпечити їхнє ефективне використання та масштабованість [23].

Визначення хмарних технологій

Хмарні технології базуються на використанні обчислювальних потужностей і ресурсів через Інтернет. Користувачі отримують доступ до цих ресурсів через веб-інтерфейси або API. Завдяки цьому бізнес та організації можуть уникнути витрат на обслуговування власного фізичного обладнання та зосередитися на основних процесах.

Хмарні сервіси поділяються на три основні категорії:

- IaaS (Infrastructure as a Service) – оренда базових обчислювальних ресурсів, таких як віртуальні машини, сховища та мережі.
- PaaS (Platform as a Service) – надання платформи для розробки та запуску додатків.
- SaaS (Software as a Service) – доступ до програмного забезпечення через Інтернет.

Переваги використання хмарних технологій

Хмарні технології мають низку переваг, що роблять їх привабливими для сучасних бізнесів, організацій та розробників.

- Масштабованість

Хмари дозволяють динамічно збільшувати або зменшувати обчислювальні ресурси відповідно до потреб. Це особливо важливо для компаній, що працюють у мінливих умовах або мають сезонний характер попиту.

- Економія коштів

Використання хмарних рішень усуває необхідність у дорогому обладнанні та його технічному обслуговуванні. Компанії платять лише за ті ресурси, які використовують, що дозволяє оптимізувати витрати.

- Доступність і мобільність

Хмарні сервіси доступні з будь-якої точки світу, де є Інтернет. Це забезпечує гнучкість для віддаленої роботи та взаємодії команд.

- Безпека даних

Провайдери хмарних послуг інвестують значні кошти у забезпечення безпеки даних. Вони використовують передові технології для захисту інформації, такі як шифрування та багаторівневий доступ.

- Автоматизація оновлень

Хмарні сервіси автоматично оновлюють програмне забезпечення та обладнання. Це зменшує навантаження на внутрішні IT-команди і забезпечує використання сучасних технологій.

- Надійність

Хмари пропонують високий рівень відмовостійкості завдяки резервуванню даних та розподілу інфраструктури. Це дозволяє забезпечити безперервність роботи додатків навіть у разі збоїв.

- Інновації

Завдяки доступу до передових інструментів і платформ, хмарні технології стимулюють швидше впровадження інновацій і скорочують час виходу продукту на ринок.

Хмарні технології стали фундаментальною частиною сучасного цифрового світу. Вони дозволяють компаніям бути більш гнучкими, економічно ефективними та конкурентоспроможними. У майбутньому їхній вплив на бізнес, державний сектор і

повсякденне життя буде лише зростати, забезпечуючи нові можливості для розвитку та впровадження інновацій.

2.1.2 Інтеграція з Azure для зберігання та обробки даних

Azure, як одна з провідних хмарних платформ від Microsoft, пропонує широкий набір сервісів для зберігання, обробки та аналізу даних. Інтеграція з Azure дозволяє забезпечити масштабованість, безпеку та високу продуктивність додатків, спрощуючи процеси управління даними [28].

Зберігання даних в Azure

Azure надає різноманітні рішення для зберігання даних, що задовольняють різні потреби:

- Azure Blob Storage

Це сервіс для зберігання неструктурованих даних, таких як зображення, відео, резервні копії та журнали. Blob Storage підтримує три типи об’єктів: блокові блоби, сторінкові блоби та блоби з додатками, забезпечуючи ефективне управління великими обсягами даних.

- Azure SQL Database

Реляційна база даних, яка працює за моделлю PaaS (платформа як сервіс). Вона забезпечує автоматичне масштабування, резервне копіювання та високу продуктивність для роботи з великими наборами даних.

- Azure Cosmos DB

Глобально розподілена NoSQL база даних, яка забезпечує масштабування та низьку затримку. Cosmos DB підтримує різні моделі даних, включаючи документи, ключ-значення, графи та стовпці.

- Azure Table Storage

Економічне рішення для зберігання структурованих даних у форматі ключ-значення. Це ідеально підходить для додатків, що потребують збереження великих обсягів простих даних.

Інтеграція з Azure для зберігання та обробки даних відкриває широкі можливості для розробників додатків. Використання Azure допомагає створювати

масштабовані, продуктивні та безпечні рішення, що відповідають сучасним потребам бізнесу та організацій.

2.1.3 Масштабованість та безпека в хмарних середовищах

Хмарні середовища стали невіддільною частиною сучасної IT-інфраструктури завдяки їхній здатності забезпечувати масштабованість та високий рівень безпеки. Вони дозволяють організаціям ефективно адаптувати ресурси до змінних потреб і забезпечують комплексний захист даних та додатків [28].

Безпека в хмарних середовищах

Безпека є важливим аспектом хмарних технологій, оскільки обробка й зберігання даних відбуваються у віддалених центрах.

Хмарні провайдери пропонують комплексний набір інструментів і політик для захисту даних і додатків:

1. Шифрування даних

Усі дані можуть бути зашифровані як під час передачі (TLS/SSL), так і під час зберігання. Наприклад, Azure підтримує шифрування за допомогою AES-256 для зберігання даних у сховищах.

2. Контроль доступу та автентифікація

За допомогою механізмів, таких як Azure Active Directory або AWS Identity and Access Management (IAM), можна створювати деталізовані політики доступу. Це дозволяє обмежувати доступ до ресурсів лише авторизованим користувачам.

3. Моніторинг і аудит

Сервіси хмарного моніторингу, як-от Azure Security Center або AWS CloudTrail, дозволяють відстежувати дії користувачів, а також виявляти потенційні загрози чи аномалії.

4. Захист від DDoS-атак

Хмарні провайдери пропонують вбудовані механізми для захисту від розподілених атак відмови в обслуговуванні (DDoS). Наприклад, Azure DDoS Protection використовує розподілену мережу для виявлення та блокування загроз.

5. Резервне копіювання та відновлення

Хмарні сервіси забезпечують регулярне резервне копіювання даних та швидке відновлення у разі збоїв або атак. Масштабованість та безпека у хмарних середовищах є ключовими факторами, які сприяють популярності цих технологій серед бізнесів. Вони забезпечують ефективну роботу навіть під великим навантаженням, мінімізуючи ризики втрати даних і підвищуючи довіру користувачів.

2.2 Безпека даних у додатках

2.2.1 Методи шифрування та їх застосування у веб і мобільних застосунках

Шифрування є основою для забезпечення безпеки у веб- і мобільних застосунках, особливо у випадках, коли необхідно захищати конфіденційні дані, такі як особисті дані користувачів, облікові записи, транзакції чи корпоративну інформацію. Воно дозволяє перетворювати відкритий текст у зашифрований, який може бути розшифрований лише за наявності відповідного ключа. Це забезпечує захист даних навіть у разі їхнього перехоплення зловмисниками [22].

У сучасних веб- та мобільних застосунках застосовуються різноманітні методи шифрування, серед яких:

1. Симетричне шифрування

Симетричне шифрування використовує один і той самий ключ для шифрування і розшифрування даних. Воно швидке і часто використовується для захисту великих обсягів інформації.

Алгоритми: AES (Advanced Encryption Standard), DES (Data Encryption Standard), 3DES.

DES (Data Encryption Standard)

Це симетричний блоковий алгоритм шифрування, який був розроблений для захисту електронних даних. Його розробка почалася у 1970-х роках, і у 1977 році DES був затверджений Національним інститутом стандартів і технологій США (NIST) як стандарт шифрування. На той час він вважався надійним і використовувався для широкого спектра застосувань у фінансовій, державній та інших сферах.

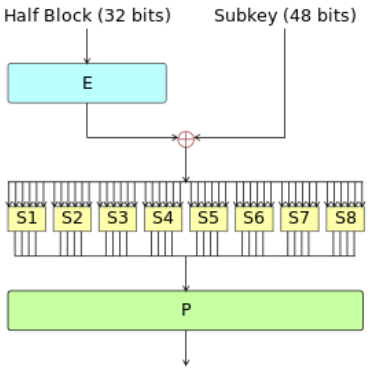


Рис. 2.1. Схема побудови алгоритму DES

Переваги DES:

- Простий і ефективний для реалізації як на програмному, так і на апаратному рівні.
- Став основою для подальшого розвитку сучасних алгоритмів шифрування.

AES (Advanced Encryption Standard)

Це один із найнадійніших і найпоширеніших алгоритмів симетричного шифрування, який використовується для захисту даних у цифрових системах. Його було розроблено як заміну алгоритму DES (Data Encryption Standard), оскільки останній став уразливим через розвиток обчислювальних потужностей. У 2001 році AES був офіційно затверджений Національним інститутом стандартів і технологій США (NIST) як стандарт шифрування.

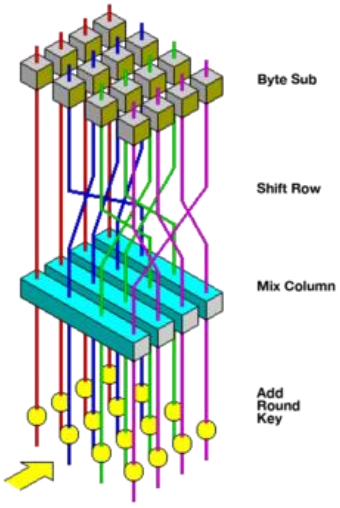


Рис. 2.2. Схема побудови алгоритму AES

Переваги AES:

- Висока продуктивність**:** алгоритм ефективно працює як на апаратному, так і на програмному рівні.
- Стійкість до атак**:** AES є надзвичайно надійним і стійким до сучасних криптографічних атак, таких як атаки грубою силою (brute-force).
- Широке застосування**:** використовується у VPN, SSL/TLS, Wi-Fi (WPA2), мобільних застосунках, базах даних тощо.

Застосування AES у веб- та мобільних додатках

Шифрування даних**:**

- Захист особистих даних користувачів, таких як імена, адреси, реквізити.
- Шифрування файлів, які передаються між клієнтом і сервером.

Захист токенів і сесій:

- використання AES для шифрування токенів доступу, які зберігаються у браузерах чи мобільних додатках.

Кешування:

- Шифрування кешованих даних на пристроях користувачів для запобігання їхньому перехопленню.

Безпечна аутентифікація:

- Шифрування паролів перед їх передачею на сервер.

Завдяки своїй гнучкості, надійності та високій продуктивності AES залишається ключовим інструментом для захисту інформації в сучасних технологіях.

2.3 Використання аутентифікації та авторизації в системах

2.3.1 Підходи до реалізації аутентифікації в веб і мобільних застосунках

Розробка проєкту для управління акаунтами з функціональністю створення та зберігання сесій є важливим аспектом забезпечення аутентифікації у веб та мобільних застосунках. Основна ідея полягає в тому, щоб створити серверну частину, яка відповідатиме за збереження даних про користувача та його сесії, а також надаватиме механізми для перевірки автентичності запитів від клієнтських додатків.

Під час розробки серверної частини використовується технологія ASP.NET Core, яка забезпечує гнучкість і надійність у реалізації сучасних систем аутентифікації. Коли користувач намагається увійти до свого облікового запису, він надсилає дані (наприклад, електронну пошту та пароль) на бекенд. Сервер перевіряє ці дані, порівнюючи збережений хеш пароля з бази даних із тим, що отриманий від користувача. У разі успішної автентифікації створюється унікальна сесія.

Сесія представляє собою запис, який включає інформацію про користувача, час початку сесії та, за необхідності, додаткові метадані, як-от IP-адресу або тип клієнта (мобільний або веб). Ця сесія зберігається в базі даних, що дозволяє забезпечити постійну доступність даних навіть у разі розподіленої архітектури. Використання бази даних для зберігання сесій забезпечує централізоване управління, зручність аналізу активності користувачів і можливість швидко завершити активну сесію у випадку виявлення підозрілої активності.

Для взаємодії з веб і мобільними застосунками сервер використовує токени. Наприклад, після створення сесії сервер може згенерувати JWT (JSON Web Token) [13], який відправляється клієнта у відповідь. Цей токен містить зашифровану інформацію про сесію і підписаний сервером, щоб клієнт міг використовувати його для авторизації наступних запитів. Таким чином, клієнтський застосунок (веб або мобільний) додає токен у заголовок кожного запиту, що дозволяє серверу перевіряти його дійсність, а також отримувати ідентифікаційну інформацію про користувача без необхідності запитувати базу даних при кожному зверненні.

Мобільний додаток і веб-додаток працюють у тісній інтеграції з сервером. Вони відповідають за збереження отриманого токена та використання його під час запитів, наприклад, для доступу до захищених ресурсів. У випадку закінчення терміну дії токена клієнт може надіслати запит на оновлення сесії, що дозволяє створити новий токен без необхідності повторного входу в обліковий запис.

Такий підхід дозволяє ефективно забезпечити аутентифікацію користувачів у розподіленій системі, що включає кілька клієнтів, при цьому залишаючись гнучким і масштабованим для адаптації до потреб зростаючого проєкту.

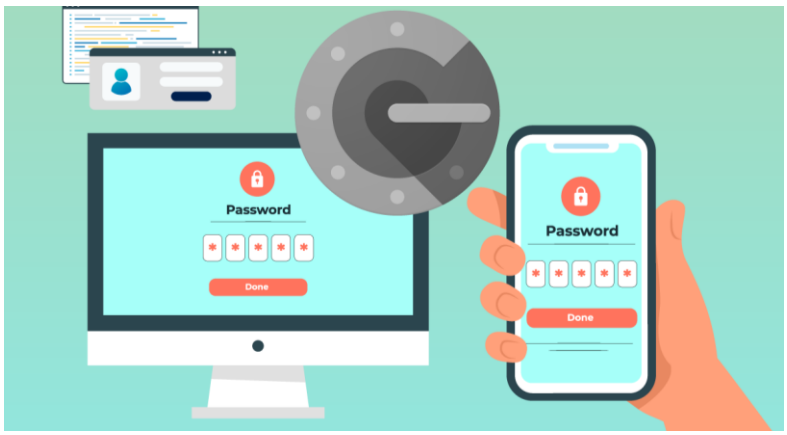


Рис 2.3. Схема взаємодії аутентифікації web і мобільного пристрою

Для забезпечення ще вищого рівня безпеки додано механізми відстеження активності сесій та їх завершення у разі виявлення підозрілих дій, таких як вхід із невідомого пристрою або зміна IP-адреси. Також реалізовано можливість перегляду активних сесій у кабінеті користувача та їх примусового завершення за потреби.

У системі використовується HTTPS для всіх запитів між клієнтом і сервером, що гарантує захист даних від перехоплення. Шифрування всього трафіку, включно з передачами токенів, забезпечує конфіденційність навіть у публічних мережах.

Таке рішення є не тільки ефективним із точки зору безпеки, але й достатньо масштабованим для роботи із зростаючою кількістю користувачів. Завдяки сучасним підходам до розробки система може бути легко адаптована до нових вимог і інтегрована з іншими сервісами, забезпечуючи повноцінний захист і зручність для користувачів.

2.3.2 Використання управління доступом і правами користувачів

Управління доступом і правами користувачів є критичним компонентом побудови сучасних програмних систем, що забезпечує захист даних, контроль над ресурсами та персоналізацію користувацького досвіду. Система управління доступом базується на розподілі прав та обов’язків між користувачами, що дозволяє забезпечити безпечний і структурований доступ до функціоналу залежно від ролі користувача в організації. У нашому випадку реалізована модель на основі ролей (Role-Based Access Control, RBAC), яка надає можливість задавати права доступу для

кожної категорії користувачів – глобальний адміністратор (Global Admin), HR-менеджер (HR) і звичайний користувач (User).

Архітектура управління доступом

Ключова ідея полягає у чіткому розмежуванні прав, щоб кожен користувач мав доступ тільки до тієї інформації та функцій, які необхідні для виконання його завдань. Це досягається шляхом створення таблиць ролей і прав (permissions), які інтегруються із системою автентифікації. Після входу в систему користувач отримує токен, що включає його роль. Серверна частина, на основі цієї ролі, виконує перевірку прав під час кожного запиту до захищених ресурсів.

Ролі та їхні права

Глобальний адміністратор (*Global Admin*)

Ця роль має найвищий рівень доступу. Глобальний адміністратор може:

- Керувати всіма аспектами системи, включно з додаванням, редагуванням і видаленням користувачів.
- Задавати нові ролі та права, оновлювати їх або видаляти.
- Мати повний доступ до всіх даних, включаючи звіти, статистику та журнал дій.
- Проводити аудит і моніторинг активності користувачів.
- Керувати конфігураціями безпеки, такими як двофакторна автентифікація або правила для сесій.

HR-менеджер (HR)

HR-менеджери виконують завдання, пов’язані з управлінням персоналом і рекрутингом. Їхні права обмежуються такими діями:

- Доступ до інформації про користувачів, які подали заявки на роботу, включаючи їхні резюме.
- Управління статусами кандидатів, наприклад, змінювати їхній прогрес у процесі рекрутингу.
- Генерація звітів про успішність набору кандидатів.
- Перегляд статистики, яка стосується найму, але без доступу до фінансових даних або конфіденційної інформації компанії.

Користувач (User)

Це основна роль для кінцевих користувачів системи. Їхні права включають:

- Доступ до особистого кабінету, де можна редагувати власну інформацію.
- Перегляд відповідних вакансій і подання заявки на них.
- Отримання повідомлень про статус поданих заявок або нові можливості.
- Обмежений доступ до загальної аналітики, якщо це дозволено системою (наприклад, тренди вакансій у певній галузі).

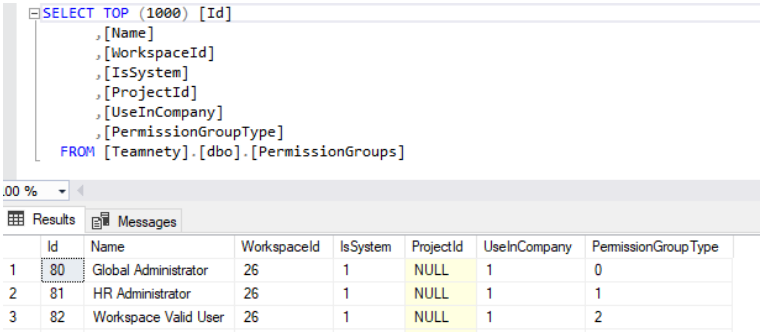
Реалізація пермішенів

У системі кожна роль пов’язана з набором дозволів (permissions), які визначають конкретні дії, доступні для цієї ролі. Наприклад, глобальний адміністратор може мати пермішени типу *ManageUsers*, *ViewAllReports*, *ModifyRoles*, тоді як HR матиме тільки *ViewApplicants* і *ManageRecruitmentStatus*.

Така структура дозволяє легко масштабувати систему: якщо потрібно додати нову роль, адміністратор просто створює її та прив’язує необхідні дозволи. Крім того, за потреби можна застосувати атрибути дозволів до окремих ресурсів.

Для реалізації пермішенів використовується багатошарова архітектура. На рівні бази даних створюються таблиці, що дозволяють зберігати інформацію про ролі, дозволи та зв’язки між ними:

- Users: таблиця, що містить інформацію про користувачів системи.
- Permission Groups: таблиця, що містить перелік усіх можливих дозволів у системі.



```
SELECT TOP (1000) [Id]
      ,[Name]
      ,[WorkspaceId]
      ,[IsSystem]
      ,[ProjectId]
      ,[UseInCompany]
      ,[PermissionGroupType]
  FROM [Teamnety].[dbo].[PermissionGroups]
```

00 %

Results | Messages

	Id	Name	WorkspaceId	IsSystem	ProjectId	UseInCompany	PermissionGroupType
1	80	Global Administrator	26	1	NULL	1	0
2	81	HR Administrator	26	1	NULL	1	1
3	82	Workspace Valid User	26	1	NULL	1	2

Рис. 2.4. Таблиця Permission Groups

- Permission Group Members: таблиця, яка пов’язує юзерів з дозволами, визначаючи, які права мають користувачі конкретної ролі.

```
SELECT TOP (1000) [Id]
      ,[ApplicationUserId]
      ,[PermissionGroupId]
  FROM [Teamnety].[dbo].[PermissionGroupMembers]
```

00 % | Results | Messages

	Id	ApplicationUserId	PermissionGroupId
1	53	a08ab782-fdbd-4e...	86
2	55	35ae637d-e982-46...	82
3	171	778c54ca-5254-48...	82

Рис 2.5. Таблиця Permission Group Members

- Permission Record: таблиця, яка визначає, які ролі призначені групою дозволів.

```
SELECT TOP (1000) [Id]
      ,[Name]
      ,[SystemName]
      ,[Category]
      ,[PermissionGroupId]
      ,[UseInCompany]
  FROM [Teamnety].[dbo].[PermissionRecord]
```

00 % | Results | Messages

	Id	Name	SystemName	Category	PermissionGroupId	UseInCompany
1	146	Manage users	ManageUsers	Standard	80	1
2	161	Manage candidates	ManageApplicants	Standard	80	1
3	162	Manage project users	ManageProjectUsers	Standard	80	0
4	163	Manage annual leave	ManageAnnualLeave	Standard	80	0
5	166	Manage leave types	ManageLeaveTypes	Standard	80	0
6	167	Manage holidays	ManageHolidays	Standard	80	0
7	171	Manage projects	ManageProjects	Standard	80	0
8	202	Manage users	ManageUsers	Standard	86	1
9	203	Manage jobs	ManageJobs	Standard	87	1
10	204	Manage candidates	ManageApplicants	Standard	87	1
11	205	Manage annual leave	ManageAnnualLeave	Standard	87	0
12	206	Manage leave	ManageLeave	Standard	87	0
13	207	Manage leave types	ManageLeaveTypes	Standard	87	0
14	208	Manage holidays	ManageHolidays	Standard	87	0
15	209	Manage employees	ManageEmployees	Standard	87	0
16	210	Manage skills	ManageSkills	Standard	87	0

Рис. 2.6. Таблиця Permission Record

На прикладному рівні, коли користувач надсилає запит на сервер, його токен, згенерований під час автентифікації, перевіряється для отримання ролей та дозволів. Потім система оцінює, чи має користувач відповідний дозвіл для виконання

конкретної дії. Наприклад, якщо HR спробує змінити конфігурації системи, сервер відхилить цей запит через відсутність відповідного дозволу в його ролі.

На рівні фронтенду застосовуються обмеження, щоб користувачі бачили лише ті функції, які відповідають їхнім ролям. Наприклад, звичайний користувач не матиме доступу до екранів управління вакансіями або резюме кандидатів.

Крім того, на серверній стороні перевірка дозволів може включати додаткові рівні контролю, такі як перевірка прав доступу до конкретних даних. Наприклад, навіть якщо користувач має право доступу до певної функції, він може мати обмежений доступ до конкретних даних на основі власних параметрів (наприклад, доступ лише до тих вакансій чи резюме, які стосуються його підрозділу або географічного регіону).

Також на сервері можуть застосовуватися політики безпеки, що визначають дозволи не тільки для окремих користувачів, а й для груп чи ролей. Це дозволяє створювати більш гнучку систему доступу, де кожна роль може мати чітко визначені функції та обмеження в залежності від необхідності і рівня доступу.

З боку фронтенду, користувачі повинні мати інтуїтивно зрозумілий інтерфейс, що надає доступ тільки до тих функцій, які відповідають їхнім ролям. Це можна досягти шляхом умовного рендерингу компонентів: коли система перевіряє роль користувача та лише виводить дозволені елементи інтерфейсу, як-от кнопки, меню або форми. Наприклад, якщо користувач не має прав доступу до певних функцій, ці елементи можуть бути приховані або заблоковані.

Додатково важливо, щоб навіть у випадку підробки запитів або маніпуляцій на клієнтській стороні, сервер все одно виконував перевірку прав доступу та відхиляв запити, які не відповідають ролі чи дозволам користувача. Це гарантує, що навіть якщо користувач спробує обійти обмеження на фронтенді, сервер зможе правильно обробити запит і забезпечити належний рівень безпеки.

Висновки до розділу

У цьому розділі детально аналізуються хмарні технології та методи забезпечення безпеки даних у сучасних IT-рішеннях. Хмарні обчислення, що базуються на наданні ресурсів через Інтернет, є основою для розробки додатків і сервісів, забезпечуючи економію ресурсів, масштабованість, доступність та високий рівень безпеки. Інтеграція хмарних технологій, таких як Microsoft Azure, дозволяє оптимізувати процеси зберігання та обробки даних завдяки використанню передових сервісів, серед яких Azure Blob Storage, Azure SQL Database та Azure Cosmos DB.

У хмарних середовищах пріоритет надається масштабованості та безпеці, що досягається завдяки шифруванню даних, контролю доступу, моніторингу активності та захисту від DDoS-атак. Важливим аспектом є використання методів шифрування, які забезпечують конфіденційність даних у веб- і мобільних додатках. Симетричне та асиметричне шифрування разом із хешуванням дозволяють гарантувати захист інформації навіть у випадку її перехоплення.

Окрім безпосередньої безпеки, важливою перевагою хмарних платформ є можливість автоматизації резервного копіювання даних, що значно знижує ризик втрати інформації. Інструменти на основі штучного інтелекту, такі як Azure Cognitive Services [28], допомагають виявляти аномалії у даних, прогнозувати можливі загрози та покращувати прийняття рішень.

Також не менш важливим є впровадження механізмів управління ідентичностями та доступом (Identity and Access Management, IAM). Використання служб, як-от Azure Active Directory, дозволяє забезпечити централізований контроль над доступом до даних і сервісів, що значно знижує ризик несанкціонованого доступу. З урахуванням стрімкого зростання обсягу даних, хмарні обчислення також пропонують інструменти для аналізу та візуалізації інформації, наприклад, Azure Data Factory або Power BI.

Усі ці технології спрямовані на створення безпечних, ефективних і інноваційних рішень, що відповідають вимогам сучасного цифрового світу.

РОЗДІЛ 3 ОСНОВНІ МОДЕЛІ, ІНТЕРФЕЙСИ ТА АЛГОРИТМИ

3.1 Основні моделі та інтерфейси системи

3.1.1 Опис основних моделей

Таблиця Resume

Ця модель описує резюме користувача на платформі пошуку роботи або в кадровій системі, яка дозволяє зберігати та управляти деталями резюме. Кожна властивість моделі має специфічне призначення, і вони разом дозволяють зберігати інформацію про досвід, кваліфікацію, статус, зарплату, тип роботи та інші аспекти профілю користувача.

```
94 references
public class Resume : Entity
{
    24 references | 0 exceptions, · live
    public string Title { get; set; }
    32 references | 0 exceptions, · live
    public int ProfileId { get; set; }
    87 references | 0 exceptions, · live
    public virtual Profile Profile { get; set; }
    [DataType(DataType.DateTime)]
    8 references | 0 exceptions, · live
    public DateTime CreatedDate { get; set; }
    [DataType(DataType.DateTime)]
    15 references | 0 exceptions, · live
    public DateTime UpdatedDate { get; set; }
    11 references | 0 exceptions, · live
    public ResumeStatuseType Status { get; set; }
    6 references | 0 exceptions, · live
    public EmploymentType? EmploymentType { get; set; }
    6 references | 0 exceptions, · live
    public RemoteWorkPrefType RemoteWorkPreferences { get; set; }
    4 references | 0 exceptions, · live
    public ExperienceYearsType? ExperienceYears { get; set; }
    2 references | 0 exceptions, · live
    public bool IsCurrentlyEmployed { get; set; }
    0 references | 0 exceptions, · live
    public bool IsIncognito { get; set; }
    5 references | 0 exceptions, · live
    public string Summary { get; set; }
    1 reference | 0 exceptions, · live
    public string AdditionalInfo { get; set; }
    8 references | 0 exceptions, · live
    public int? Salary { get; set; }
    15 references | 0 exceptions, · live
    public int? IndustryId { get; set; }
    9 references | 0 exceptions, · live
    public virtual Industry Industry { get; set; }
    2 references | 0 exceptions, · live
    public CurrencyType CurrencyType { get; set; }
    14 references | 0 exceptions, · live
    public int Rating { get; set; }
    11 references | 0 exceptions, · live
    public bool IsDraft { get; set; }
    3 references | 0 exceptions, · live
    public int? PictureId { get; set; }
    19 references | 0 exceptions, · live
    public virtual Picture Picture { get; set; }
```

Рис. 3.1. Таблиця Resume

Ось детальний опис кожної з властивостей:

Title – Назва резюме. Це поле може бути використано для короткого опису або назви позиції, на яку претендує кандидат (наприклад, "Senior Software Engineer").

ProfileId – Унікальний ідентифікатор профілю користувача, до якого належить це резюме. Це дозволяє зв’язати резюме з конкретним користувачем.

Profile – Навігаційне властивість, що вказує на об’єкт типу Profile, який містить додаткову інформацію про користувача, таку як ім'я, контактні дані тощо.

CreatedDate – Дата та час створення резюме. Це поле допомагає відслідковувати, коли було створене резюме.

UpdatedDate – Дата та час останнього оновлення резюме. Вказує на момент, коли резюме було змінено останній раз.

Status – Статус резюме, що визначає, чи є резюме активним, на розгляді або закритим. Це може бути пов'язано з певними етапами перегляду або публікації.

EmploymentType – Тип зайнятості, який вказує на бажану форму роботи (повна зайнятість, часткова, контракт, тощо). Це поле може бути порожнім, якщо користувач не вказав тип зайнятості.

RemoteWorkPreferences – Вказує на перевагу щодо віддаленої роботи (наприклад, чи кандидат готовий працювати віддалено або тільки в офісі).

ExperienceYears – Кількість років досвіду, яку має кандидат у відповідній сфері.

IsCurrentlyEmployed – Логічне значення (true/false), яке вказує, чи є кандидат зараз працевлаштованим.

IsIncognito – Логічне значення (true/false), яке вказує, чи є резюме анонімним (якщо true, інформація про кандидата є непублічною).

Summary – Короткий опис досвіду, кваліфікацій та цілей кандидата.

AdditionalInfo – Додаткова інформація про кандидата, яку він хоче вказати в резюме (можливо, з додатковими навичками чи досягненнями).

Salary – Очікувана заробітна плата кандидата. Це поле може бути порожнім, якщо кандидат не вказав очікування.

IndustryId – Ідентифікатор галузі, до якої належить резюме (наприклад, IT, маркетинг, фінанси).

Industry – Навігаційне властивість, яке вказує на об’єкт Industry, що містить назву галузі.

CurrencyType – Валюта, в якій вказана зарплата (наприклад, USD, EUR, UAH).

Rating – Рейтинг кандидата, що може відображати загальну оцінку або відгуки від роботодавців.

IsDraft – Логічне значення (true/false), яке вказує, чи є резюме чернеткою і ще не опубліковане для загального доступу.

Picture – Навігаційне властивість, яке вказує на об’єкт Picture, що містить фото профілю кандидата.

Ця модель дозволяє створити гнучку систему управління резюме, що підтримує різні типи інформації, від статусу резюме до фотографії кандидата, типу роботи та багато іншого. Всі ці властивості можна використовувати для створення детальних профілів кандидатів, з якими можуть працювати роботодавці на платформі.

Таблиця ApplicationUser

Ця модель описує користувача в системі, який може бути частиною компанії або особистим користувачем на платформі. Модель зберігає дані про ім’я користувача, його активність, позицію в організації, зв'язок з профілем, зворотний зв'язок та налаштування сповіщень.

```
99+ references
public class ApplicationUser : IdentityUser, IEntity<string>
{
    99+ references | 0 exceptions, - live
    public string FirstName { get; set; }
    99+ references | 0 exceptions, - live
    public string LastName { get; set; }
    16 references | 0 exceptions, - live
    public bool IsRemoved { get; set; }
    6 references | 0 exceptions, - live
    public bool IsActive { get; set; } = true;
    9 references | 0 exceptions, - live
    public string Position { get; set; }
    10 references | 0 exceptions, - live
    public bool IsCompanyUser { get; set; }
    8 references | 0 exceptions, - live
    public bool IsOnline { get; set; }
    - references | 0 exceptions, - live
    public virtual UserProfile.Profile Profile { get; set; }
    1 reference | 0 exceptions, - live
    public virtual ICollection<FeedBack> FeedBacks { get; set; }
    8 references | 0 exceptions, - live
    public EmailNotificationSetting EmailNotification { get; set; }

    [DataType(DataType.DateTime)]
    1 reference | 0 exceptions, - live
    public DateTime CreatedDate { get; set; }
```

Рис. 3.2. Таблиця Application User

Ось детальний опис кожної з властивостей:

FirstName – Ім’я користувача. Це текстове поле зберігає перше ім’я користувача в системі.

LastName – Прізвище користувача. Це поле зберігає прізвище користувача.

IsRemoved – Логічне значення (true/false), яке вказує, чи було видалено користувача з системи (наприклад, якщо обліковий запис був деактивований або видалений).

IsActive – Логічне значення (true/false), яке вказує, чи є користувач активним в системі. За замовчуванням встановлено в значення true, що означає, що користувач активний.

Position – Позиція користувача в компанії або організації. Це текстове поле може містити роль або звання користувача, наприклад, "Менеджер проєкту", "Розробник" тощо.

IsCompanyUser – Логічне значення (true/false), яке вказує, чи є користувач частиною компанії. Якщо значення true, користувач належить до певної організації, і його акаунт може мати додаткові дозволи та функції для співробітників компанії.

IsOnline – Логічне значення (true/false), яке вказує, чи є користувач в даний момент онлайн. Це поле може бути корисним для відображення статусу користувача в реальному часі.

Profile – Навігаційне властивість, яке вказує на об’єкт типу UserProfile.Profile, що містить додаткову інформацію про користувача, таку як контактні дані, фотографії, біографія та інші параметри, що описують користувача.

FeedBacks – Колекція об’єктів типу FeedBack, яка містить відгуки про користувача. Це дозволяє зберігати відгуки, оцінки або коментарі від інших користувачів або адміністраторів, що можуть бути пов’язані з діяльністю цього користувача.

EmailNotification – Об'єкт типу EmailNotificationSetting, який містить налаштування для сповіщень користувача, що надсилаються електронною поштою. Це може включати налаштування для отримання різних сповіщень, таких як сповіщення про нові вакансії, оновлення профілю тощо.

CreatedDate – Дата та час створення облікового запису користувача в системі. Це поле фіксує момент, коли користувач був зареєстрований або створений в системі.

Ця модель дозволяє ефективно керувати користувачами на платформі, надаючи всю необхідну інформацію для ідентифікації користувача, налаштувань сповіщень, зворотного зв'язку та стану облікового запису. Вона підтримує різні аспекти користувача, від його основних даних (ім’я, прізвище) до статусу активності та налаштувань. Модель також дозволяє зберігати відгуки користувачів і забезпечує можливість зміни налаштувань сповіщень через електронну пошту.

Таблиця Device

Ця модель представляє пристрій в системі, ймовірно, для використання в контексті платформи, що має справу з різними пристроями, наприклад, для отримання статистики або взаємодії з користувачами через мобільні чи інші типи пристроїв.

```
– references
public class Device : Entity, IEntity<int>
{
    24 references | 0 exceptions, - live
    public string DeviceSecret { get; set; }
    42 references | 0 exceptions, - live
    public string Token { get; set; }
    2 references | 0 exceptions, - live
    public string Manufacturer { get; set; }
    2 references | 0 exceptions, - live
    public string Product { get; set; }
    2 references | 0 exceptions, - live
    public string Model { get; set; }
    2 references | 0 exceptions, - live
    public string DeviceName { get; set; }
    2 references | 0 exceptions, - live
    public string Brand { get; set; }
    – references | 0 exceptions, - live
    public DateTime DateTime { get; set; }
    1 reference | 0 exceptions, - live
    public DateTime? UpdatedDateTime { get; set; }
    10 references | 0 exceptions, - live
    public DevicePlatform DevicePlatform { get; set; }
    4 references | 0 exceptions, - live
    public DeviceCompany DeviceCompany { get; set; }
    4 references | 0 exceptions, - live
    public bool AllowNotifications { get; set; }
    10 references | 0 exceptions, - live
    public bool IsDeleted { get; set; } = false;
    0 references | 0 exceptions, - live
    public string IP { get; set; }
```

Рис. 3.3. Таблиця Device

Ось детальний опис кожної з властивостей цієї моделі:

DeviceSecret – Унікальний ідентифікатор для пристрою. Це значення може використовуватись для автентифікації пристрою в системі або як ключ для взаємодії з іншими компонентами платформи.

Token – Токен, що може бути використаний для аутентифікації або авторизації пристрою. Зазвичай це поле пов’язане з механізмами безпеки, де токен може бути використаний для забезпечення доступу до специфічних ресурсів або API.

Manufacturer – Виробник пристрою. Ця властивість може містити назву компанії, що виробляє пристрій, наприклад, "Apple", "Samsung" тощо.

Product – Продукт, до якого відноситься пристрій. Це поле може містити загальну назву продукту, як наприклад "iPhone" чи "Galaxy".

Model – Модель пристрою. Це текстове поле містить конкретну модель пристрою, наприклад, "iPhone 13", "Samsung Galaxy S21".

DeviceName – Назва пристрою. Може бути задана користувачем або автоматичною системою для ідентифікації пристрою, наприклад, "Мій телефон" або "Телефон робітника".

Brand – Бренд пристрою. Це може бути, наприклад, "Apple", "Samsung", "Google", що ідентифікує торгову марку пристрою.

DateTime – Дата та час, коли пристрій було зареєстровано в системі. Це поле фіксує момент створення або підключення пристрою до системи.

UpdatedDateTime – Дата та час останнього оновлення інформації про пристрій. Це поле допомагає відслідковувати зміни або оновлення статусу пристрою в системі.

DevicePlatform – Платформа пристрою (наприклад, Android, iOS, Windows). Це зв'язує пристрій з конкретною операційною системою, що дозволяє системі адаптуватися до специфічних вимог або поведінки різних платформ.

AllowNotifications – Логічне значення, що вказує, чи дозволяє пристрій отримувати сповіщення від системи. Якщо значення true, то пристрій може отримувати сповіщення через push-повідомлення або інші механізми.

IsDeleted – Логічне значення, яке вказує, чи був пристрій видалений з системи. За замовчуванням це значення встановлено в false, що означає, що пристрій активний в системі.

IP – IP-адреса пристрою. Це поле містить мережеву адресу пристрою, яка може бути використана для ідентифікації пристрою в мережі або для безпеки (наприклад, для контролю доступу за IP).

Модель Device вказує на пристрій, що може бути зареєстрований в системі, з різноманітними атрибутами, що описують його характеристики (виробник, модель, бренд), а також статуси, пов'язані з його використанням (дозвіл на сповіщення, статус видалення). Вона містить як технічні дані (тип пристрою, IP-адреса), так і бізнес-дані, що описують взаємодію пристрою з користувачем або компанією.

Таблиця Job

Ця модель описує вакансію в системі, яка є частиною більшої платформи для пошуку роботи або управління вакансіями. Клас Job є сутністю, що містить різні властивості, які визначають, яку вакансію користувач може переглядати чи на яку може подати заявку.

```
99+ references
public class Job : Entity
{
    [Required]
    41 references | 0 exceptions, - live
    public string Title { get; set; }
    13 references | 0 exceptions, - live
    public JobStatus Status { get; set; }
    9 references | 0 exceptions, - live
    public JobReviewStatus ReviewStatus { get; set; }
    14 references | 0 exceptions, - live
    public EmploymentType? EmploymentType { get; set; }
    1 reference | 0 exceptions, - live
    public CurrencyType CurrencyType { get; set; }
    13 references | 0 exceptions, - live
    public SeniorityLevel? SeniorityLevel { get; set; }
    14 references | 0 exceptions, - live
    public JobType? Type { get; set; }

    33 references | 0 exceptions, - live
    public Location Location { get; set; }
    14 references | 0 exceptions, - live
    public int LocationId { get; set; }

    [Required]
    16 references | 0 exceptions, - live
    public int CompanyId { get; set; }
```

Рис. 3.4. Таблиця Job

Ось розбір кожного з полів цієї моделі:

Title – Обов'язкове текстове поле, яке містить назву вакансії. Це поле є основним для ідентифікації вакансії, наприклад, "Менеджер з продажу" або "Розробник програмного забезпечення".

Status – Перелічуване значення типу JobStatus, що визначає поточний статус вакансії, наприклад, чи вона активна, закрита, на розгляді тощо. Це поле дозволяє системі керувати різними станами вакансії.

ReviewStatus – Перелічуване значення типу JobReviewStatus, яке вказує на статус перегляду вакансії. Це може бути корисно для модерації або процесу перевірки контенту вакансії перед публікацією.

EmploymentType – Тип зайнятості, наприклад, повна зайнятість, неповна зайнятість, тимчасова чи контрактна робота. Це поле визначає, чи є вакансія на постійну роботу, чи тимчасову.

CurrencyType – Валюта, що використовується для опису заробітної плати в вакансії. Це може бути валютний тип, як-от долар, євро чи інші.

SeniorityLevel – Перелічуваний тип SeniorityLevel, що описує рівень досвіду або кваліфікації, необхідний для цієї вакансії, наприклад, початковий, середній чи старший рівень.

Type – Тип вакансії (наприклад, постійна або тимчасова). Це дозволяє уточнити, чи є вакансія, наприклад, проєктною або сезонною.

Location – Об'єкт, що описує географічне місце розташування вакансії. Це може бути місто, країна, район чи конкретна адреса, де розташоване робоче місце.

LocationId – Ідентифікатор місця розташування. Це поле допомагає зберігати посилання на запис у базі даних, що містить інформацію про місцезнаходження вакансії, для оптимізації зберігання та доступу.

CompanyId – Ідентифікатор компанії, яка публікує вакансію. Це обов'язкове поле вказує на те, яка компанія є роботодавцем для цієї вакансії.

Модель Job визначає вакансію в контексті платформи для пошуку роботи або управління вакансіями. Вона включає основні атрибути вакансії, такі як назва, статус, тип зайнятості, валюта, місце розташування та інші дані, що дозволяють чітко

визначити, які умови роботи пропонує роботодавець, та сприяє відповідному фільтруванню або пошуку вакансій. Всі ці дані дають змогу компаніям керувати своїми вакансіями та забезпечувати кандидатам зручний пошук і подачу заявок на робочі місця.

3.1.2 Розробка UI/UX з урахуванням принципів доступності та зручності користування

Розробка інтерфейсу для сторінок реєстрації та логування є важливим елементом будь-якого сучасного веб або мобільного додатку. Саме ці екрани часто стають першою взаємодією користувача із системою, тому вони повинні бути простими, інтуїтивними зрозумілими і доступними для всіх категорій користувачів. При створенні таких інтерфейсів важливо враховувати як основи дизайну, так і принципи доступності, щоб забезпечити комфортне користування для людей з різними потребами. Основні аспекти зручності користування (Usability):

Мінімалізм і структурованість

- Екрани мають виглядати чисто та зрозуміло. Використовується мінімум елементів, які спрямовують користувача до завершення одного завдання – створення акаунта.
- Розташування полів та елементів повинно бути логічним: зверху вниз, із чітким виділенням основних блоків.
- Мінімальний набір полів: ім'я**,** електронна пошта**,** пароль**,** підтвердження пароля. Це допомагає уникнути перевантаження і спрощує процес для користувача.
- Чітка навігація і зворотній зв'язок
- Користувач має отримувати миттєвий зворотній зв'язок про свої дії. Наприклад, при введенні слабкого пароля система відразу повідомляє про це повідомленням на кшталт: "Пароль має містити не менше 8 символів, включати цифри та спеціальні символи".
- Помилки, як-от порожнє поле або некоректний формат email, мають підсвічуватись червоним кольором із поясненням.

Інтуїтивність

- Використання зрозумілих іконок: наприклад, значок замка для пароля чи ока для показу/приховування пароля.
- Текстові підказки (placeholders) в полях, які зникають після введення даних. Наприклад: "Введіть свою електронну пошту".
- Кнопка реєстрації (Call to Action)
- Кнопка "Зареєструватися" повинна бути чітко виділена кольором і розташована в зоні досяжності, особливо на мобільних пристроях. Важливо забезпечити її неактивність, поки всі поля не будуть заповнені коректно.

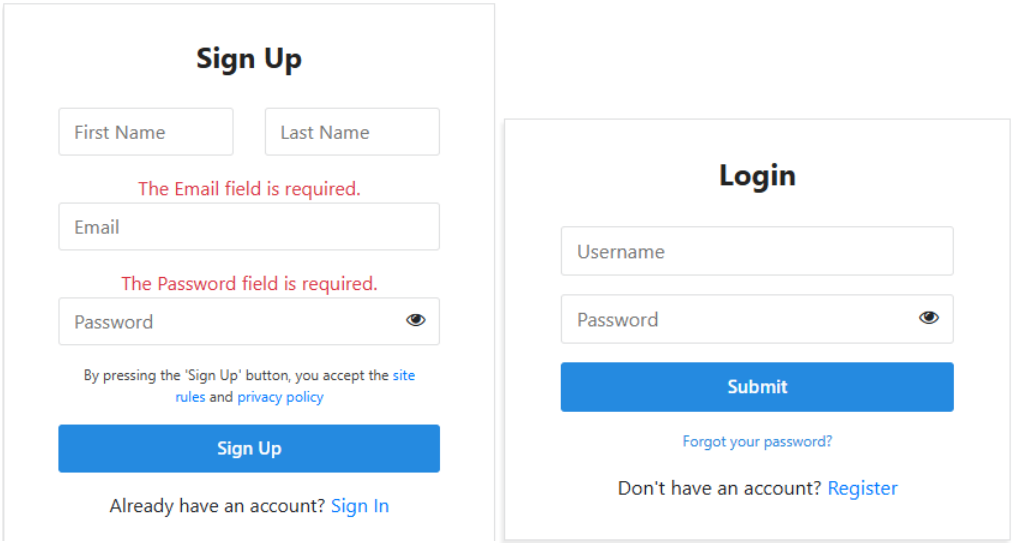


Рис. 3.5. Вигляд реєстрації

Сторінка вакансій має забезпечувати зручний і інтуїтивний доступ до інформації про робочі місця. У її основі лежить логіка, що дозволяє користувачам легко знаходити вакансії, фільтрувати їх за різними параметрами та звертати увагу на пріоритетні пропозиції, такі як гарячі вакансії.

На сторінці відображається список вакансій, кожна з яких містить назву посади, назву компанії, її логотип, локацію, зарплатний діапазон (якщо вказаний), позначки типу «Дистанційна робота» чи «Гаряча вакансія», короткий опис і кнопку для перегляду детальної інформації. Користувач може вибирати серед вакансій за допомогою інтерактивних фільтрів. Фільтри дозволяють вказати ключове слово,

локацію, тип зайнятості, рівень зарплати, досвід роботи чи категорію вакансії. При кожній зміні параметрів виконується динамічний запит до сервера, що оновлює результати пошуку без необхідності перезавантаження сторінки.

Гарячі вакансії виділяються окремим блоком, що розташований або у правій частині сторінки, або у верхній її частині. Вони позначені візуально — наприклад, червоним бейджем або спеціальним фоном. Це вакансії з підвищеною зарплатою чи ті, які мають високий пріоритет для роботодавців.

Сторінка має також зручну систему відображення статусу результатів. Якщо пошук не дав результатів, користувач отримує повідомлення з порадою змінити фільтри. У разі великої кількості результатів передбачена пагінація або безкінечний скролінг, щоб користувач міг легко переглядати вакансії.

Важливим аспектом є інтерактивність сторінки: користувач може зберігати вакансії у вибране, порівнювати декілька вакансій за ключовими параметрами або підписатися на оновлення за вибраними фільтрами.

Загалом, сторінка створена так, щоб користувач міг швидко знайти вакансію, яка відповідає його потребам, і водночас отримати приємний досвід користування завдяки простому, але функціональному інтерфейсу.

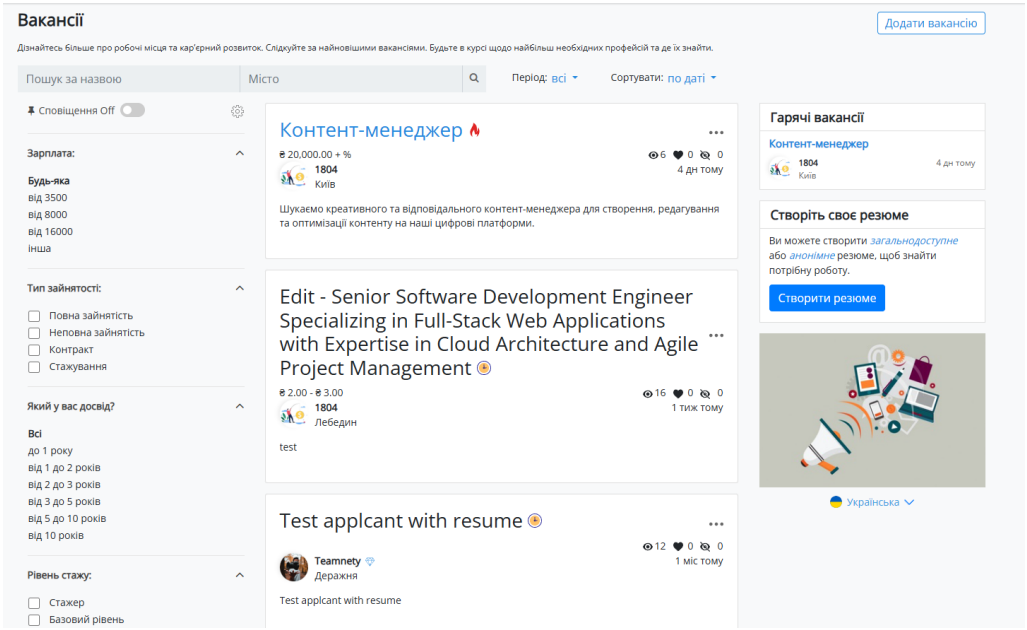


Рис. 3.6. Сторінка списку вакансій з фільтрами і пошуками

Сторінка з резюме призначена для перегляду, створення, редагування та управління резюме користувача. Її логіка орієнтована на зручність роботи з ключовими розділами, такими як особисті дані, професійні навички, досвід роботи, освіта, сертифікати та інше.

На сторінці відображається основна інформація користувача: його ім'я, фото (за наявності), контактні дані, а також короткий опис або резюме професійного досвіду. Цей блок розташований у верхній частині сторінки, щоб роботодавці могли швидко отримати загальне уявлення про кандидата.

Основна частина сторінки розділена на тематичні секції:

- Особисті дані – містить контактну інформацію (телефон, email, адреса), лінки на соціальні мережі або портфоліо.
- Професійний досвід – кожна позиція роботи відображається у вигляді картки з назвою посади, періодом роботи, назвою компанії, описом обов’язків і досягнень.
- Освіта – секція, де вказуються навчальні заклади, отримані ступені та періоди навчання.
- Навички та компетенції – інтерактивний список навичок із можливістю додати нові чи видалити зайві.
- Сертифікати – окремий розділ для завантаження або відображення сертифікатів, дипломів і рекомендаційних листів.
- Додаткова інформація – тут користувач може додати інформацію про хобі, мови чи інші деталі, що можуть зацікавити роботодавця.

Сторінка з резюме також дозволяє:

- Завантажити резюме в PDF – кнопка для експорту оформленого резюме в популярний формат.
- Переглядати стан заповнення – індикатор прогресу показує, наскільки повним є резюме.
- Режим попереднього перегляду – функція, яка дозволяє подивитися, як резюме виглядатиме для роботодавців.

Для зручності передбачені підказки та рекомендації, наприклад, як заповнити блок досвіду роботи або що додати для більшої привабливості резюме.

Якщо користувач ще не має заповненого резюме, сторінка пропонує майстер створення, який покроково допомагає додати потрібну інформацію. У мобільній версії інтерфейс адаптовано для зручності: секції відкриваються в окремих вкладках або списках, щоб уникнути перевантаження екрану.

Ця сторінка створена для того, щоб користувач міг легко створити професійне резюме, яке буде вигідно презентувати його навички й досвід потенційним роботодавцям.

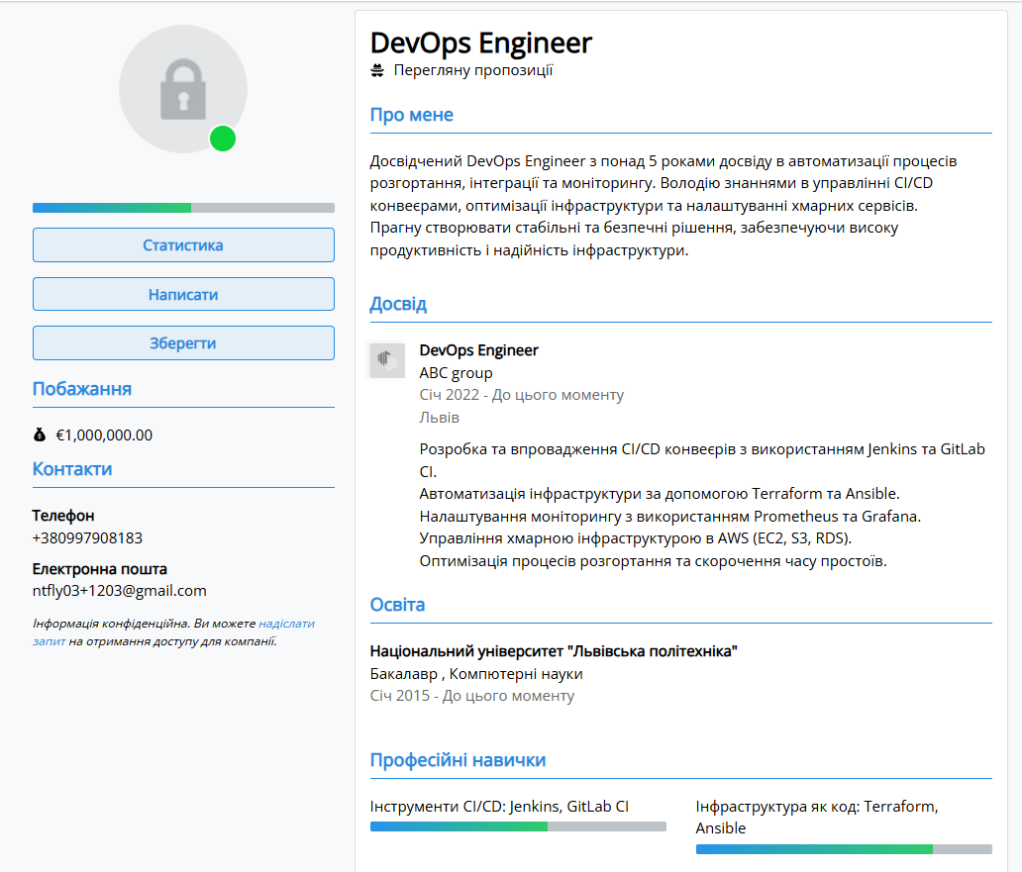


Рис. 3.7. Форма для заповнення резюме

Сторінка компанії з вкладками забезпечує комплексний доступ до різних аспектів діяльності компанії та її взаємодії з користувачами. Вкладки створені для зручної навігації та розділення функціональності, щоб користувачі могли швидко знайти потрібну інформацію або виконати необхідну дію.

Основна логіка сторінки з вкладками:

- Новини

Ця вкладка містить актуальні новини компанії, наприклад, оновлення продуктів, досягнення, зміни в організації, або майбутні події. Контент представлений у вигляді списку публікацій із датами, заголовками та коротким описом.

- Про нас

Тут розміщена детальна інформація про компанію: її місія, візія, історія, структура, ключові цінності та керівництво. Візуально вкладка може включати інфографіку, фотографії команди та відеоматеріали.

- Вакансії

Основний функціонал цієї вкладки — відображення списку вакансій, доступних у компанії. Кожна вакансія представлена у вигляді картки із заголовком, ключовими вимогами, умовами роботи та зарплатою. Також передбачені фільтри за посадою, типом зайнятості та місцем роботи.

- Рахунки

Вкладка дозволяє компанії та адміністраторам переглядати фінансову інформацію, наприклад, виставлені рахунки, історію платежів, статус оплати та плани підписок. Для користувачів передбачено завантаження рахунків або перегляд підписки на послуги.

- Продукти

У цьому розділі компанія презентує свої основні продукти чи послуги. Кожен продукт відображений із назвою, описом, ціною, фотографією або іншими візуальними матеріалами.

- Історія продуктів

Ця вкладка дозволяє користувачам відстежувати розвиток та оновлення продуктів компанії. Наприклад, зміни функціоналу, випуск нових версій або завершення підтримки старих продуктів.

- Кандидати

Тут компанія може переглядати профілі кандидатів, які подали заявки на вакансії. Вкладка включає деталі, як-от резюме, портфоліо, контактну інформацію та статус розгляду заявки.

- Резюме

У цій вкладці зберігається база резюме, надісланих кандидатами. Адміністратори можуть виконувати пошук і фільтрувати резюме за ключовими словами, досвідом роботи, освітою тощо.

- Користувачі

Ця вкладка дає доступ до списку користувачів, які мають доступ до платформи компанії. Включає інформацію про ролі, статус активності, історію входів і дій.

- Дозволи

Ця вкладка забезпечує управління ролями та доступами. Адміністратор може надавати чи обмежувати доступ до певних функцій для різних категорій користувачів.

Ця структура вкладок дозволяє компанії ефективно управляти своєю інформацією, забезпечуючи зручність та організованість у використанні платформи. Інтерфейс вкладок розроблено з урахуванням інтуїтивного дизайну, що дозволяє користувачам легко знаходити потрібну інформацію.

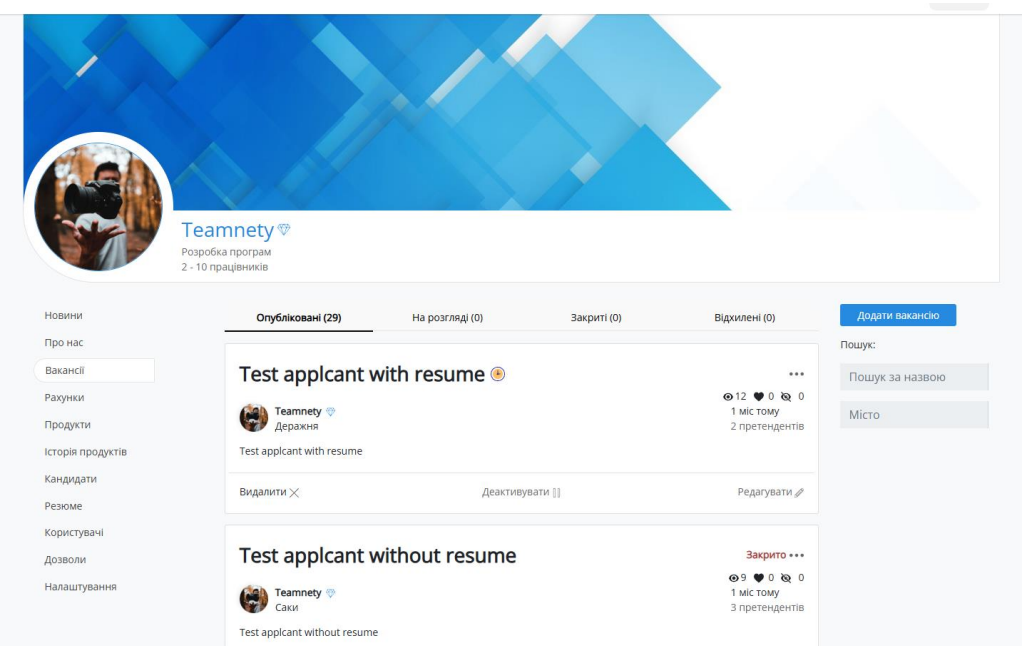


Рис. 3.8. Сторінка компанії

Така таблиця є фундаментальним інструментом для адміністраторів, що дозволяє забезпечити високу якість вакансій, представлених на платформі, і гарантувати, що вони відповідають усім необхідним стандартам. Інтуїтивно зрозумілий інтерфейс, інтеграція з іншими модулями системи та можливість гнучкого управління процесом модерації роблять її зручною та ефективною для повсякденної роботи адміністративної команди.

Таблиця для адміністратора зі статусами вакансій розроблена для ефективного управління оголошеннями про роботу, які розміщуються на платформі. Основна мета – забезпечити інструмент для перевірки відповідності вакансій політикам та критеріям сайту, а також оперативно керувати їх статусами.

Колонки таблиці:

Таблиця складається з кількох колонок, кожна з яких відповідає за ключову інформацію про вакансію:

- ID вакансії: Унікальний ідентифікатор вакансії для внутрішнього використання.
- Назва вакансії: Заголовок вакансії, що відображає її суть.
- Компанія: Назва компанії, яка подала вакансію, з можливістю перегляду її профілю.
- Дата публікації: Час, коли вакансія була створена або відправлена на модерацію.
- Статус: Поточний статус вакансії (наприклад, "Очікує на перевірку", "Схвалена", "Відхилена").
- Опис вакансії: Короткий опис з ключовими вимогами та умовами роботи, що дозволяє адміністратору швидко оцінити вакансію.
- Дії: Набір дій для зміни статусу вакансії або перегляду її деталей.

Фільтри:

Для зручності адміністратор може застосовувати фільтри, щоб переглядати вакансії за певними критеріями:

- Статуси (очікує перевірки, схвалена, відхилена).
- Компанія, яка розмістила вакансію.

- Дата створення або публікації.

Дії з вакансіями:

У колонці "Дії" представлені кнопки або випадаючий список з можливими діями:

- Перегляд деталей: Відкриття повної інформації про вакансію, включаючи повний опис, вимоги, умови роботи та додаткові файли, якщо є.
- Схвалення: Зміна статусу вакансії на "Схвалена" з повідомленням компанії.
- Відхилення: Вказівка причин відмови (наприклад, невідповідність умовам сайту або некоректний опис).

Автоматизація:

Для прискорення роботи адміністратори можуть використовувати функцію "Масових дій". Наприклад, відразу схвалити або відхилити кілька вакансій після перегляду.

Є можливість автоматичних підказок: якщо вакансія має порушення (наприклад, занадто короткий опис або неправдиві дані), система позначає її червоним кольором або попередженням.

Валідація:

- Для перевірки відповідності вакансії критеріям сайту, у вкладеному розділі може бути доступна інформація про:
- Правильність заповнення ключових полів.
- Використання заборонених слів або термінів.
- Відповідність умов праці заявленим політикам сайту.

Така таблиця є первинним інструментом для адміністраторів, що дозволяє забезпечити високу якість вакансій, представлених на платформі, і гарантувати, що вони відповідають усім необхідним стандартам. Інтуїтивно зрозумілий інтерфейс, інтеграція з іншими модулями системи та можливість гнучкого управління процесом модерації роблять її зручною та ефективною для повсякденної роботи адміністративної команди.

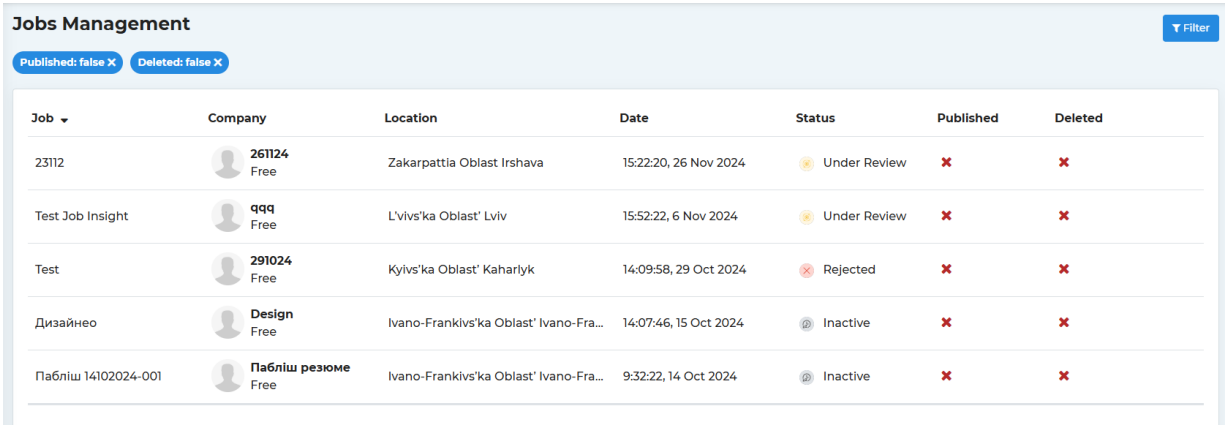


Рис. 3.9. Вигляд сторінки списку новостворених вакансій для адмін-частини

Сторінка Job Insight Management призначена для аналізу активності користувачів на платформі в контексті перегляду вакансій. На цій сторінці відображаються два основних елементи: графік з кількістю переглядів вакансій по платформам і таблиця, що показує, з яких пристроїв відбуваються перегляди.

Графік переглядів вакансій по платформам є важливим інструментом для оцінки ефективності різних каналів залучення користувачів. Графік може бути представлений у вигляді стовпчикової або лінійної діаграми, де по осі X відображається час (наприклад, за днями, тижнями чи місяцями), а по осі Y — кількість переглядів вакансій. Різні платформи — наприклад, веб, мобільні додатки або мобільні версії браузерів — можуть бути представлені різними кольорами або лініями на графіку, що дозволяє швидко зрозуміти, яка платформа має найбільший попит серед користувачів. Це допомагає зрозуміти, які канали використовуються найбільше і де необхідно проводити оптимізацію для підвищення ефективності.

Таблиця з інформацією про пристрої надає детальнішу інформацію щодо типу пристроїв, які використовуються для перегляду вакансій. У цій таблиці можуть бути вказані такі колонки:

- Тип пристрою: смартфон, планшет, десктоп.
- Платформа: мобільний додаток, веб-сайт, мобільна версія.
- Кількість переглядів: загальна кількість переглядів вакансій для кожного типу пристрою.

- Відсоток від загальної кількості: частка кожного типу пристрою від загальної кількості переглядів вакансій.

Ця таблиця дозволяє адміністраторам швидко зрозуміти, які пристрої найпопулярніші серед користувачів для перегляду вакансій, що може бути корисно для подальшої оптимізації інтерфейсу або розробки нових функцій, орієнтуючись на найбільш використовувані пристрої та платформи.

Інтерактивні фільтри, що включають вибір періоду (наприклад, поточний місяць, тиждень, рік), дозволяють динамічно змінювати графік і таблицю, щоб вивести дані на основі певного проміжку часу. Це робить сторінку ще зручнішою для користувачів, оскільки вони можуть отримати більш точні та актуальні дані для аналізу активності на платформі.

Таким чином, поєднання графіка і таблиці на сторінці Job Insight Management дозволяє отримати всебічну картину щодо переглядів вакансій і дає змогу робити аналітичні висновки для покращення залучення користувачів та оптимізації роботи платформи.

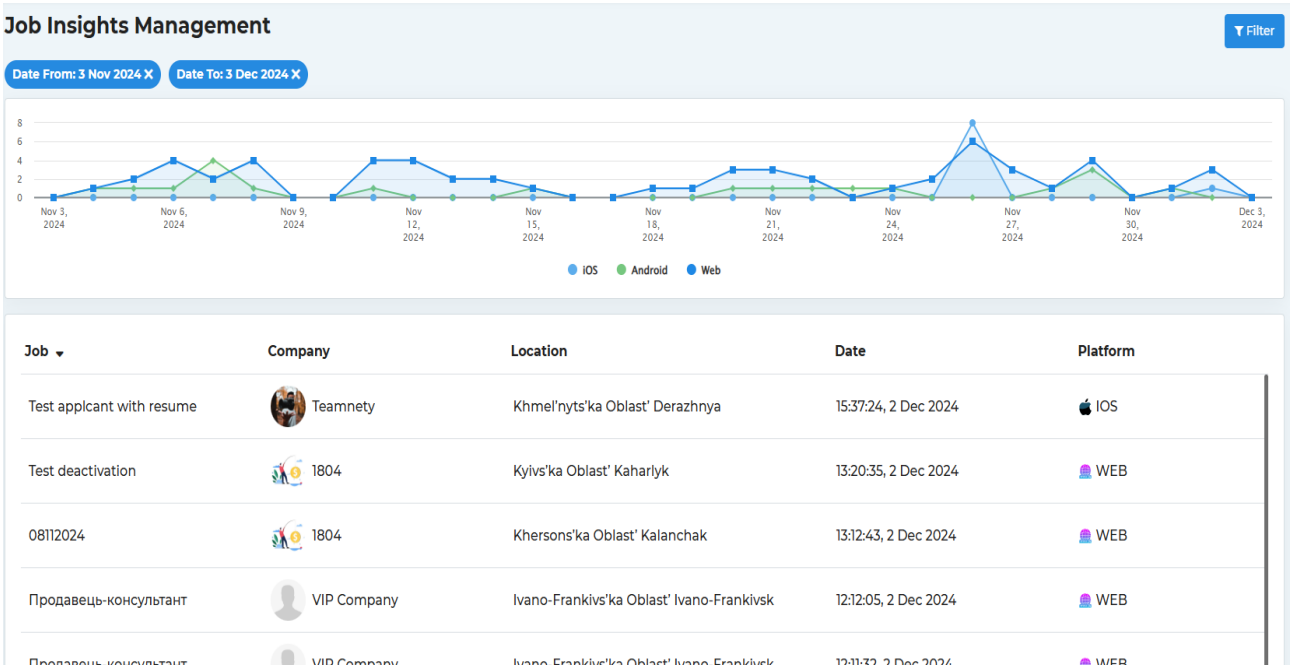


Рис. 3.10. Сторінка аналітики списку новостворених вакансій для адмін-частини

3.2 Реалізація основних функціональних алгоритмів

3.2.1 Алгоритм обробки даних користувача в системі

Алгоритм розміщення резюме

Алгоритм розміщення резюме на сайті починається з того, що користувач проходить процедуру аутентифікації. Він може увійти до системи через наявний обліковий запис або створити новий, вказавши базову інформацію, таку як ім'я, email та пароль. Після успішного входу користувач потрапляє до особистого кабінету, де йому пропонується функція створення нового резюме.

На етапі створення резюме користувач заповнює запропоновану форму. Вона містить поля для введення ключової інформації, зокрема назви резюме, досвіду роботи, навичок, бажаної зарплати, типу зайнятості, формату роботи (дистанційна чи офісна) та інших даних, що характеризують кандидата. Користувач також може додати короткий опис або мотиваційний лист, який пояснює його кар’єрні цілі та досягнення. Додатково є можливість прикріпити фотографію, вибрати шаблон оформлення або позначити резюме як чернетку, якщо воно ще не готове до публікації.

Після заповнення даних користувач надсилає резюме на збереження. Система перевіряє дані на коректність, наприклад, чи заповнені обов’язкові поля, чи відповідають введені значення допустимим форматам, та зберігає їх у базі даних. У випадку успішного збереження резюме отримує статус «Чернетка» або «Опубліковане» в залежності від вибору користувача.

Якщо користувач вирішує опублікувати резюме, система додатково перевіряє його відповідність критеріям платформи. Наприклад, визначає, чи містить резюме достатньо інформації для роботодавців. Після перевірки резюме стає доступним для перегляду роботодавцями на платформі. При цьому система може відображати інформацію про кількість переглядів, дозволяти користувачеві редагувати дані або змінювати статус резюме (наприклад, перевести його в режим «Інкогніто», коли ім’я користувача приховано).

Алгоритм також враховує опції для фільтрування резюме за типом зайнятості, досвідом роботи чи іншими параметрами, що дозволяє роботодавцям легко знаходити

відповідних кандидатів. Завершальним етапом є інтеграція з іншими функціями системи, наприклад, сповіщеннями про нові перегляди або можливістю відгуку на вакансії безпосередньо через сайт.

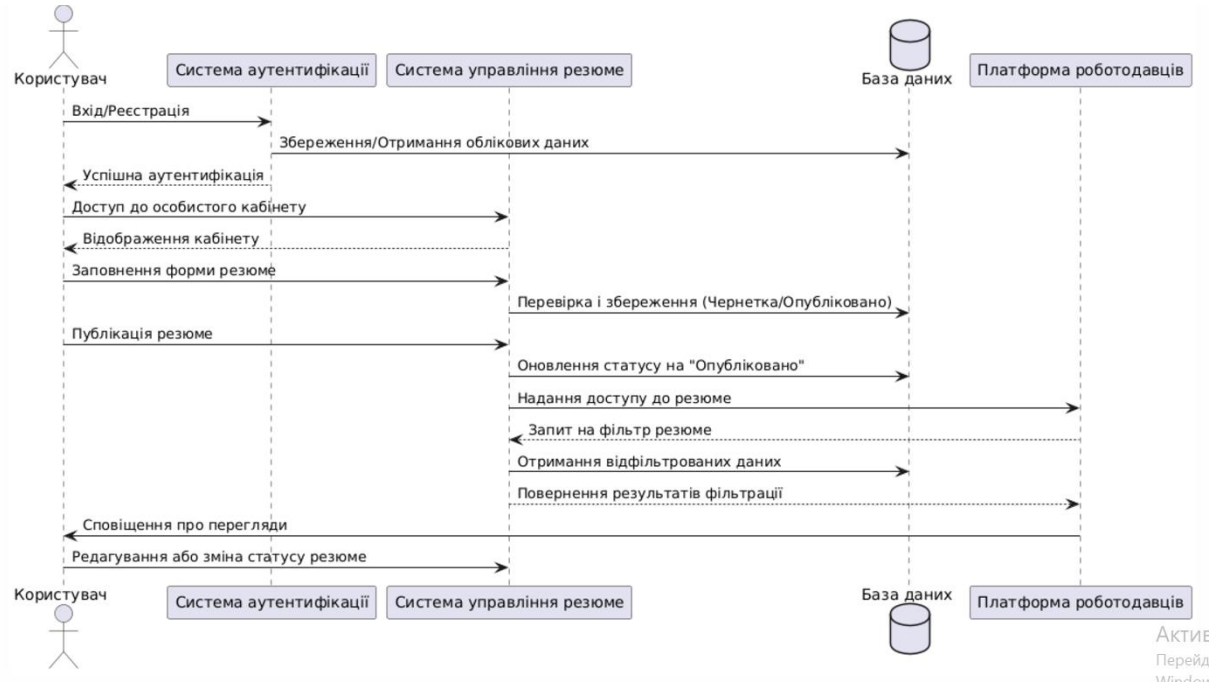


Рис. 3.11. UML діаграма послідовності алгоритму розміщення резюме

Алгоритм розміщення вакансії

Для адміністратора створеної компанії починається з того, що адміністратор проходить аутентифікацію в системі, використовуючи свої облікові дані. Після успішного входу він переходить до панелі управління компанією, яка надає доступ до функцій управління контентом, зокрема розміщення вакансій.

Адміністратор вибирає опцію "Створити нову вакансію" та переходить до форми, де вводить основну інформацію про вакансію. Це включає назву вакансії, опис обов’язків, вимог до кандидата, бажаний досвід роботи, рівень кваліфікації, тип зайнятості, місце розташування роботи та рівень заробітної плати. Додатково вказуються дані про те, чи доступна дистанційна робота, валюта заробітної плати та інформація про те, чи є вакансія пріоритетною (гарячою).

Після заповнення форми адміністратор має можливість прикріпити додаткові файли, такі як графічні матеріали чи презентації про компанію, та зазначити теги, що

допоможуть кандидатам швидше знайти вакансію через систему фільтрації. Перед відправленням вакансії система перевіряє всі поля на коректність, зокрема чи заповнені обов’язкові дані, такі як назва вакансії та опис, і чи відповідають введені значення визначеним форматам.

Після перевірки даних адміністратор може вибрати статус публікації вакансії. Якщо статус встановлено як «Чернетка», вакансія зберігається в системі, але не відображається користувачам. Якщо обрано статус «На перевірці», вакансія автоматично надсилається на модерацію. Модератор перевіряє відповідність контенту вимогам сайту, зокрема відсутність порушень правил публікації, та ухвалює рішення про її публікацію.

Після схвалення вакансія стає видимою для користувачів сайту. Вона відображається в загальному списку вакансій та доступна через фільтри за галуззю, місцем роботи, типом зайнятості тощо. Адміністратор отримує сповіщення про успішну публікацію, а також можливість переглядати статистику переглядів вакансії, кількість відгуків та інформацію про зацікавлених кандидатів. У разі потреби адміністратор може редагувати опубліковану вакансію або змінювати її статус.

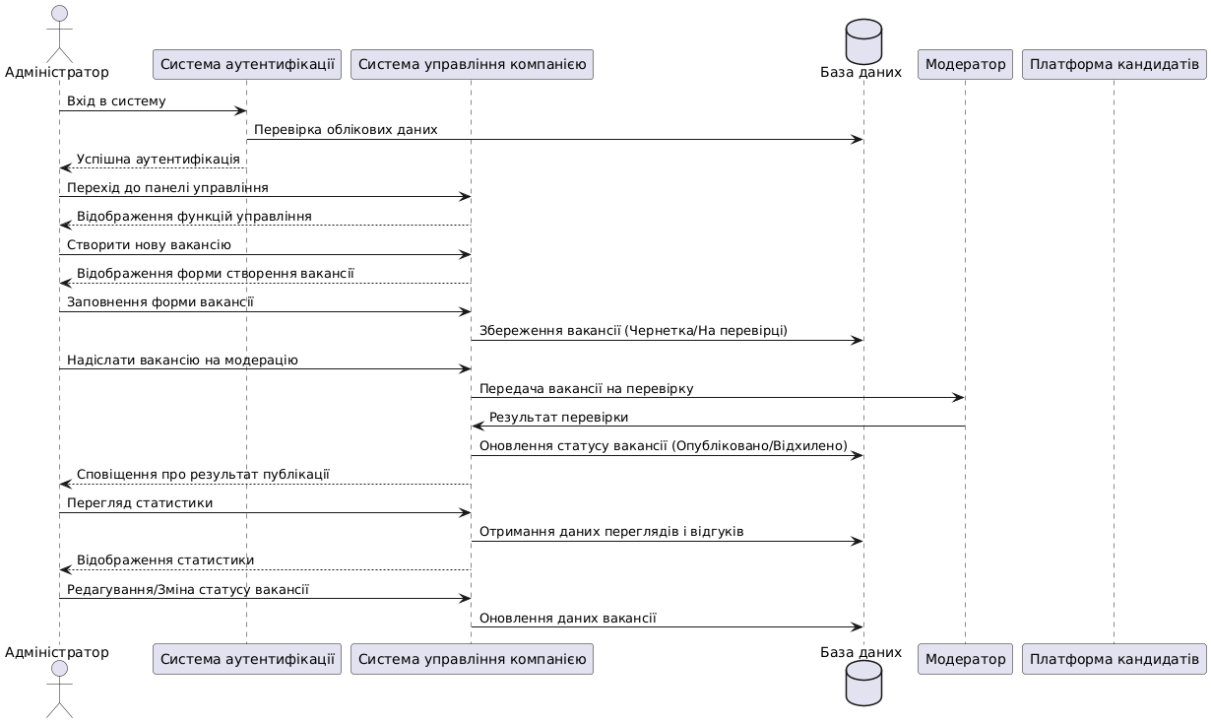


Рис. 3.12. UML діаграма послідовності алгоритму розміщення вакансії

Алгоритм обробки нотифікацій та пуш-сповіщень для адміністраторів

У випадку, коли кандидати подають резюме на вакансії, побудований для забезпечення своєчасного інформування про важливі події. Коли кандидат відправляє резюме на вакансію, система миттєво обробляє цю подію через серверний модуль. Сервер приймає запит, перевіряє його на валідність та верифікує пов’язані дані, такі як статус вакансії, ідентифікатор адміністратора компанії та відповідність поданого резюме вимогам вакансії.

Після успішної верифікації система генерує нотифікацію для адміністратора. Ця нотифікація зберігається у базі даних разом із деталями події, такими як ім'я кандидата, його резюме, назва вакансії та час подання. Одночасно система перевіряє налаштування сповіщень адміністратора, щоб визначити, чи дозволено надсилання пуш-сповіщень.

Якщо пуш-сповіщення увімкнено, система надсилає повідомлення на зареєстрований пристрій адміністратора через відповідний сервіс, наприклад Firebase Cloud Messaging. Повідомлення містить короткий опис події, посилання для перегляду деталей та опції для подальших дій, наприклад, переглянути резюме чи зв'язатися з кандидатом.

Адміністратор, отримавши сповіщення, може перейти на відповідний розділ адміністративної панелі, щоб переглянути всі нові заявки. У панелі зібрані всі нотифікації, включно з тими, які адміністратор отримав раніше. Кожна нотифікація має статус (наприклад, «нове», «переглянуто») і автоматично оновлюється після взаємодії адміністратора з нею.

У разі, якщо адміністратор не реагує на сповіщення, система може повторно надіслати нагадування через визначений інтервал часу. Це гарантує, що важливі події не залишаться без уваги. Такий підхід дозволяє підтримувати ефективну комунікацію між системою та адміністраторами, сприяючи швидкому обробленню заявок кандидатів і поліпшенню загального користувацького досвіду.

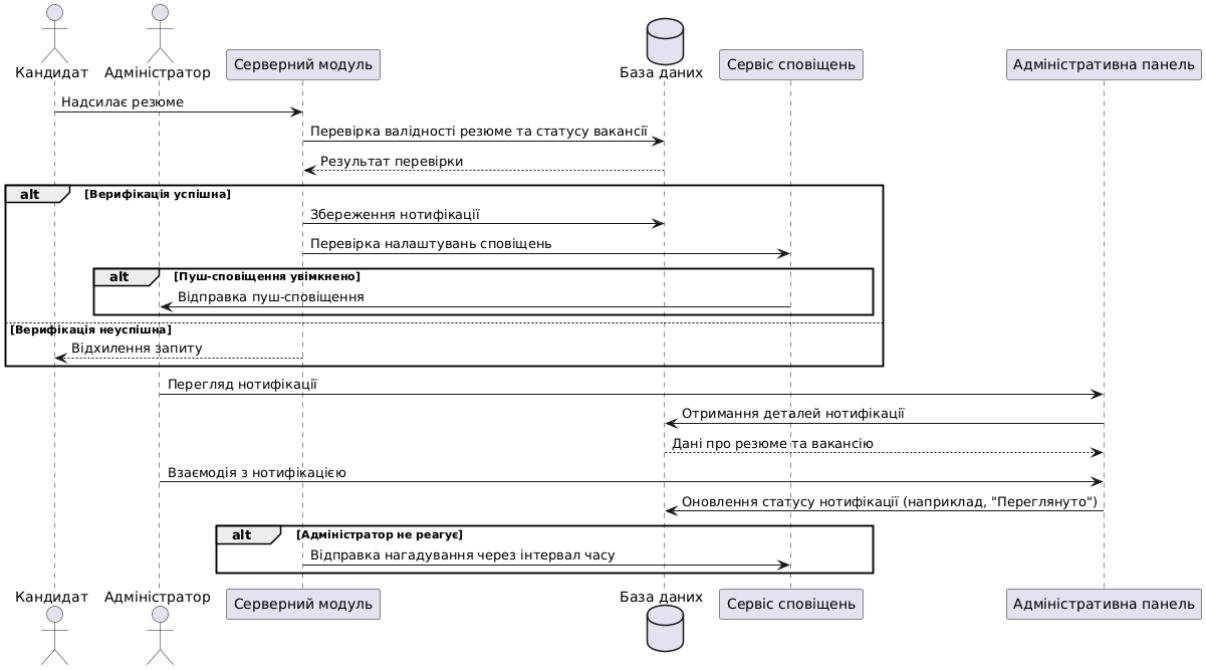


Рис. 3.13. UML діаграма послідовності алгоритму обробки нотифікацій та пуш-сповіщень для адміністраторів

Алгоритм обирання темплейту

З 33 запропонованих шаблонів у системі є можливість створювати особливий вигляд резюме. Користувач, який хоче вибрати шаблон для свого резюме або іншого документа, спочатку отримує доступ до списку з 33 доступних варіантів через інтерфейс користувача. Цей список може бути представлений у вигляді прев'ю або мініатюр, що дозволяє переглядати зовнішній вигляд кожного шаблону. Кожен шаблон має відповідну назву та опис, що допомагає користувачу зрозуміти, який тип шаблону найбільше відповідає його вимогам.

Після перегляду доступних шаблонів користувач може застосувати фільтри або сортування для звуження вибору за категоріями, такими як стиль, формат, чи інші атрибути. Користувач обирає потрібний шаблон, натискаючи на нього. Після цього система відображає більше деталей щодо вибраного шаблону, дозволяючи користувачу оцінити, чи відповідає він його вимогам.

Після того, як користувач вирішує, який шаблон йому підходить, він натискає на кнопку підтвердження вибору, і система зберігає цей шаблон для подальшого використання. Вибір шаблону може також бути збережений у профілі користувача

або безпосередньо прив'язаний до конкретного резюме чи документа, залежно від функціоналу платформи.

В результаті цього процесу, користувач обирає найбільш підходящий шаблон серед 33 варіантів, що забезпечує інтуїтивно зрозумілий і зручний інтерфейс для взаємодії з системою.

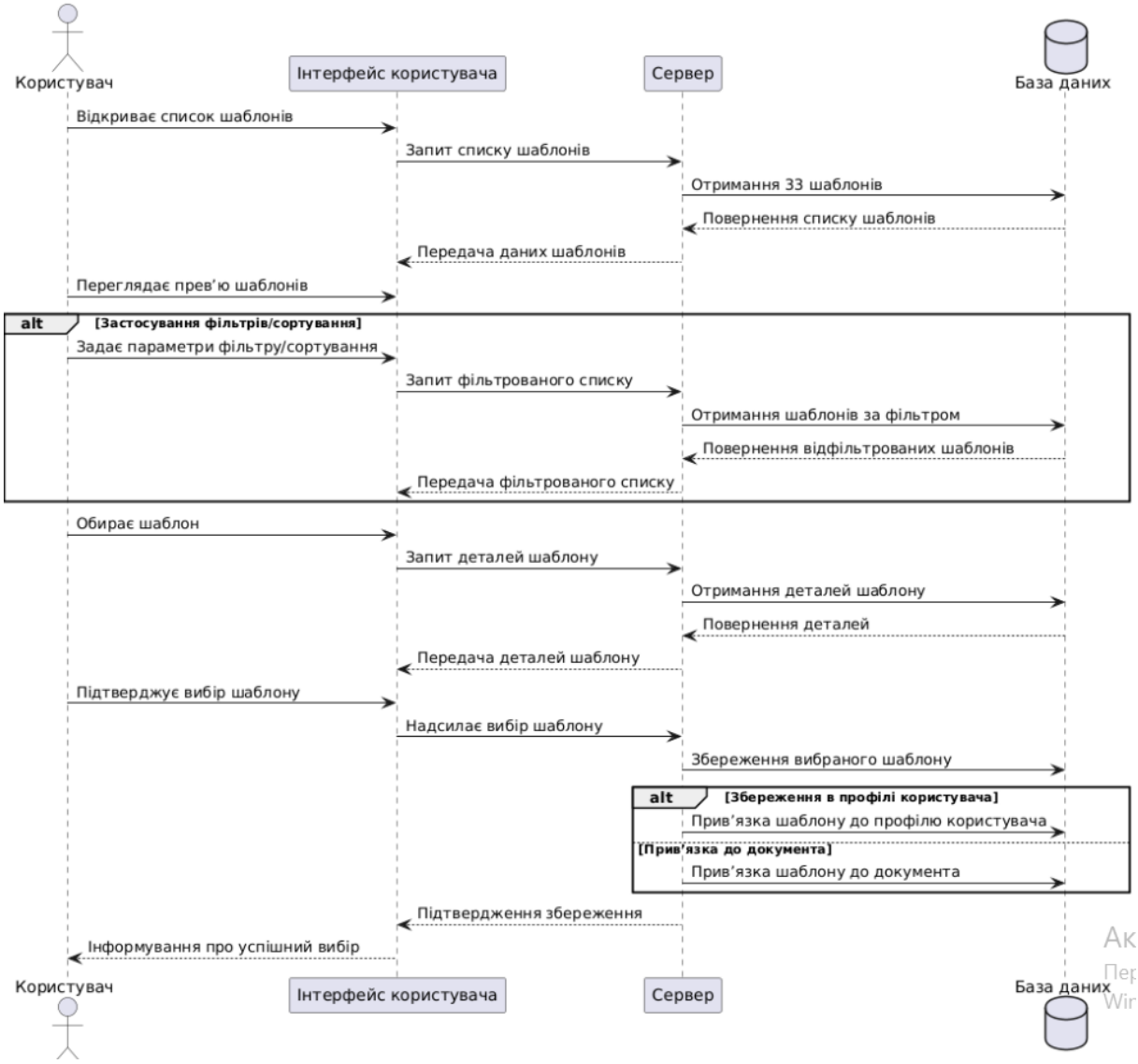


Рис. 3.14. UML діаграма послідовності алгоритму обирання темплейту

3.2.2 Створення механізмів взаємодії між сервером та клієнтом

Створення механізмів взаємодії між сервером та клієнтськими додатками є суттєвим аспектом розробки сучасних програмних систем. Така взаємодія забезпечує передачу даних, команд та запитів між різними компонентами програми,

забезпечуючи їхню інтеграцію та узгоджену роботу. Основою для цього є архітектурні рішення, що використовуються для зв’язку серверів та клієнтів.

Основні елементи та підходи до взаємодії:

- API (Application Programming Interface): сервер надає чітко визначений набір функцій у вигляді API, які клієнтські додатки можуть викликати для отримання або передачі даних. RESTful API, GraphQL або gRPC є популярними підходами до реалізації таких інтерфейсів.
- Протоколи передачі даних: найпоширенішим протоколом є HTTP/HTTPS, який забезпечує передачу даних у структурованих форматах, таких як JSON або XML. У реальному часі можуть використовуватися WebSocket або протоколи передачі потоків даних, такі як SSE (Server-Sent Events) [15].
- Автентифікація та авторизація: взаємодія між сервером і клієнтськими додатками потребує надійної автентифікації користувачів і забезпечення доступу лише до дозволених ресурсів. Це реалізується за допомогою токенів доступу (наприклад, JWT), OAuth2, OpenID Connect або інших технологій.
- Обмін даними у реальному часі: для задач, які вимагають моментального оновлення інформації (наприклад, чати, нотифікації, моніторинг даних), використовуються WebSocket або хмарні сервіси повідомлень, такі як Firebase Cloud Messaging (FCM).
- Кешування даних: щоб зменшити навантаження на сервер і пришвидшити взаємодію, використовуються механізми кешування, як-от Redis або вбудоване кешування на стороні клієнта за допомогою IndexedDB [11].
- Обробка асинхронних подій: для складних систем, які працюють із великою кількістю клієнтських запитів, впроваджуються черги повідомлень (наприклад, RabbitMQ, Kafka), що забезпечують надійну обробку запитів у фоновому режимі.
- Безпека передачі даних: дані між сервером і клієнтом повинні шифруватися, щоб уникнути їх перехоплення. HTTPS використовується для шифрування, а також можуть впроваджуватися додаткові механізми, такі як підписані запити чи валідація даних на стороні сервера.

- Реалізація версіонування API: для підтримки сумісності між різними версіями клієнтів і сервера впроваджується версіонування API, що дозволяє оновлювати серверну логіку, не порушуючи функціонал існуючих клієнтів.

Приклад взаємодії:

- Клієнтський додаток надсилає запит до API сервера (наприклад, для отримання списку вакансій).
- Сервер обробляє запит, перевіряє автентифікацію, виконує запит до бази даних і повертає відповідь у форматі JSON.
- На стороні клієнта отримані дані відображаються у відповідному UI.
- У випадку події (наприклад, оновлення статусу вакансії) сервер ініціює push-сповіщення для клієнта через WebSocket.

3.2.3 Розробка та інтеграція API для комунікації між модулями

Розробка та інтеграція API з використанням ASP.NET Core та SQL Server забезпечує ефективну комунікацію між модулями системи, дотримуючись принципів чистої архітектури.

База даних SQL Server використовується для зберігання та управління даними, а Entity Framework Core спрощує доступ до них. RESTful API виступає інтерфейсом для взаємодії, де контролери обробляють HTTP-запити, а бізнес-логіка організована в сервісах, що реалізують інтерфейси. Впровадження маппінгу між об'єктами, як-от AutoMapper, забезпечує спрощене перетворення між моделями та DTO.

Такий підхід гарантує масштабованість, модульність і зручність підтримки системи, дозволяючи легко інтегрувати нові функціональні модулі та адаптувати систему до змін.

Основні компоненти

Чиста архітектура:

- Core (Domain Layer): включає бізнес-логіку, сутності, інтерфейси та специфікації.

- Application Layer: реалізує інтерфейси, керує взаємодією між доменом і зовнішніми модулями. Тут знаходяться сервіси, DTO, мапери (наприклад, AutoMapper).
- Infrastructure Layer: реалізує роботу з базою даних (SQL Server), зовнішніми API, чергами повідомлень тощо.
- Presentation Layer (API): відповідає за прийом і обробку HTTP-запитів.

База даних (SQL Server):

- Використовується Entity Framework Core для зручної роботи з SQL Server.
- Усі моделі й зв’язки налаштовуються у DbContext.
- ASP.NET Core API:
- RESTful API як стандарт взаємодії.
- Swagger для автоматичної документації.
- Впровадження Dependency Injection для керування залежностями.

Етапи реалізації API для комунікації між модулями

1. Ініціалізація проєкту

Створено новий проєкт за допомогою шаблону ASP.NET Core Web API. Створено папки для шарів: *Domain, Application, Infrastructure, API*.

2. Конфігурація бази даних

У Infrastructure створено ApplicationDbContext, який успадковує DbContext.

Налаштовано підключення до SQL Server у `appsettings.json`

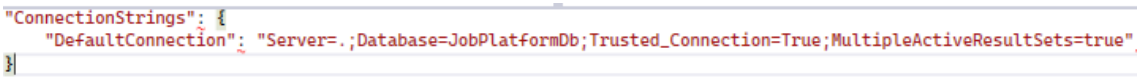


Рис. 3.15. Налаштування Connection Strings

Зареєстровано контекст у `Startup.cs` або `Program.cs`

```
services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
```

Рис. 3.16. Налаштування SQL service

Розробка доменної моделі (Domain Layer)

У шарі Domain створіть основні моделі, наприклад:

```
99+ references
public class Job : Entity
{
    [Required]
    41 references | 0 exceptions, · live
    public string Title { get; set; }
    13 references | 0 exceptions, · live
    public JobStatus Status { get; set; }
    9 references | 0 exceptions, · live
    public JobReviewStatus ReviewStatus { get; set; }
    14 references | 0 exceptions, · live
    public EmploymentType? EmploymentType { get; set; }
    1 reference | 0 exceptions, · live
    public CurrencyType CurrencyType { get; set; }
    13 references | 0 exceptions, · live
    public SeniorityLevel? SeniorityLevel { get; set; }
    14 references | 0 exceptions, · live
    public JobType? Type { get; set; }

    33 references | 0 exceptions, · live
    public Location Location { get; set; }
    14 references | 0 exceptions, · live
    public int LocationId { get; set; }

    [Required]
    16 references | 0 exceptions, · live
    public int CompanyId { get; set; }
```

Рис. 3.17. Entity Job у шарі домену

Реалізація бізнес-логіки (Application Layer)

Додано інтерфейси:

- Реалізовано їх у відповідних сервісах.
- Використано AutoMapper для перетворення між моделями й DTO

```
CreateMap<Job, JobPublicDetailsDto>()
    .ForMember(x => x.Company, config => { config.MapFrom(x =>
        new CompanyJobDto
        {
            Name = x.Company.Name,
            imageUrl = x.Company.ImageUrl,
            CoverUrl = x.Company.CoverUrl,
            CompanySize = x.Company.CompanySize,
            JobCount = x.Company.Jobs.Count(key => key.CompanyId == x.Id && key.Status == JobStatus.Published && !key.IsInActive),
            CompanyPageType = x.Company.CompanyPageType,
            IsCompanyMessage = x.Company.IsCompanyMessage
        });
    })
    .ForMember(x => x.PictureUrl, config => { config.MapFrom(p => p.JobPictures.OrderByDescending(pp => pp.Id).Select(pp => pp.Picture.AltAttribute).FirstOrDefault()); })
    .ForMember(x => x.ProfileId, config => { config.MapFrom(x => x.CreatedBy.Profile.Id); })
    .ForMember(x => x.Location, config => { config.MapFrom(x => x.Location.Name); })
    .ForMember(x => x.Industry, config => { config.MapFrom(x => x.Company.Industry.Name); })
    .ForMember(x => x.IndustryId, config => { config.MapFrom(x => x.Company.Industry.Id); })
    .ForMember(x => x.CompanyOwnerProfileId, config => { config.MapFrom(x => x.Company.Workspace.Owner.Profile.Id); })
    .ForMember(x => x.Statistic, config => { config.MapFrom(x =>
        new JobStatisticDto
        {
            ViewCount = x.ViewCount,
            SavedCount = x.SavedCount,
            HiddenCount = x.HiddenCount
        });
    })
```

Рис. 3.18. Приклад мапінгу даних між Job та JobPublicDetailsDto

Розробка контролера (Presentation Layer)

У API створено контролер для обробки запитів:

```
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;
using Teamnety.Services.Jobs;

[ApiController]
[Route("api/[controller]")]
1 reference
public class JobsController : ControllerBase
{
    private readonly IJobService _jobService;

    0 references | 0 exceptions, - live
    public JobsController(IJobService jobService) => _jobService = jobService;

    [HttpGet]
    1 reference | 0 requests, - live | 0 exceptions, - live
    public async Task<IActionResult> GetAllJobs() =>
        Ok(await _jobService.GetAllJobsAsync());

    [HttpPost]
    0 references | 0 requests, - live | 0 exceptions, - live
    public async Task<IActionResult> CreateJob(JobDto job)
    {
        var jobId = await _jobService.CreateJobAsync(job);
        return CreatedAtAction(nameof(GetAllJobs), new { id = jobId }, job);
    }
}
```

Рис. 3.19. Приклад Job Controller

Висновки до розділу

У розділі даному розділі було детально розглянуто створення ключових компонентів платформи для пошуку роботи. Зокрема, було розроблено інтерфейси, що забезпечують інтуїтивно зрозумілу взаємодію користувачів із системою. Вони адаптовані до різних пристроїв і потреб користувачів, дозволяючи легко здійснювати пошук вакансій, заповнювати резюме, а також отримувати персоналізовані рекомендації.

Описані алгоритми включають персоналізований підбір вакансій, автоматизовану оцінку резюме, обробку користувацьких запитів у реальному часі та алгоритм аналітики для роботодавців. Кожен алгоритм оптимізовано для ефективного використання ресурсів системи та інтегровано в загальну архітектуру платформи.

В результаті розробки вдалося забезпечити високий рівень автоматизації та персоналізації процесів, що підвищує ефективність взаємодії між роботодавцями та кандидатами, а також сприяє покращенню користувацького досвіду. Успішна інтеграція інтерфейсів і алгоритмів доводить перспективність реалізованого підходу до створення інноваційної платформи.

ВИСНОВКИ

У даній роботі було розглянуто побудову моделей, методів та алгоритмів для інноваційних платформ пошуку роботи, які забезпечують ефективну комунікацію між роботодавцями та кандидатами. Особлива увага приділялася сучасним рішенням, спрямованим на персоналізацію процесу пошуку вакансій і резюме, оптимізацію взаємодії користувачів із системою.

В ході дослідження було виявлено, що успішна реалізація таких платформ потребує ретельного підходу до організації обміну даними між модулями, забезпечення конфіденційності інформації, створення механізмів аутентифікації та авторизації. У роботі акцентувалося на розробці алгоритмів персоналізованого підбору вакансій, які аналізують відповідність резюме вимогам роботодавців, а також моделей управління вакансіями, що дозволяють зручно створювати, перевіряти та оновлювати позиції. Це включає інтеграцію з базами даних для збереження резюме, вакансій та профілів користувачів, а також забезпечення зручних методів пошуку та фільтрації інформації.

Також було розглянуто важливість забезпечення безпеки та конфіденційності даних користувачів, а також використання механізмів аутентифікації та авторизації для захисту інформації. Було створено та вдосконалено моделі управління вакансіями, які дозволяють адміністраторам зручно створювати нові позиції, перевіряти їх на відповідність і оперативно інформувати користувачів про нові можливості. Розроблені алгоритми для генерації та доставки сповіщень забезпечують своєчасне інформування адміністраторів про важливі події, такі як подання резюме кандидатами.

Представлені алгоритми генерації та доставки сповіщень, спрямовані на оперативне інформування про важливі події, такі як подання резюме. Отримані результати підтверджують ефективність запропонованих рішень і демонструють значний потенціал для впровадження персоналізованих, автоматизованих інструментів на платформі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шевчук, І. (2018). Інформаційні технології в системах пошуку роботи: методи та алгоритми. Київ: Наукова думка.
2. Кучеренко, В. (2020). Інноваційні платформи для пошуку роботи: аналіз сучасних підходів та технологій. Журнал "Технічні науки", 45(3), 125-136.
3. Державна служба зайнятості України (2023). Онлайн-платформи для пошуку роботи та працівників. Офіційний сайт. https://www.dcz.gov.ua/
4. Міністерство освіти і науки України. (2022). Використання штучного інтелекту в кадрових системах та підвищення ефективності найму персоналу. https://mon.gov.ua/
5. Кабінет Міністрів України (2023). Аналіз ситуації на ринку праці: нові технології та підходи. Офіційний портал. https://www.kmu.gov.ua/
6. Міжнародна конференція з інновацій в сфері кадрових технологій. (2021). Системи штучного інтелекту для підбору персоналу в Україні. https://innovations.jobtech.org.ua/
7. Закон України "Про електронні документи та електронний документообіг" (2022). https://zakon.rada.gov.ua/
8. Національний технічний університет України "Київський політехнічний інститут". (2020). Моделі та алгоритми для автоматичного підбору персоналу. http://kpi.ua/
9. Шаблони та алгоритми для підбору персоналу на основі великої кількості даних. (2022). Журнал "Технології управління персоналом", 28(5), 78-89.
10. Системи автоматизованого підбору персоналу на основі аналітики великих даних. (2021). Аналітичний огляд наукових статей, 33(6), 152-161.
11. Державний портал відкритих даних України. (2023). Інтеграція відкритих даних у платформи пошуку роботи. https://data.gov.ua/
12. Борисенко, О. (2020). Розробка моделей машинного навчання для оптимізації підбору персоналу в Україні. Журнал "Інформаційні технології", 11(4), 90-102.

13. Medium. Блог TypeScript Weekly. (2023). https://typescript-weekly.com/

14. TypeScript Deep Dive by Basarat Ali Syed. Книга для глибокого вивчення TypeScript. (2023). https://basarat.gitbook.io/typescript/

15. Технічні рішення для поліпшення процесу пошуку роботи через автоматизацію процесів. (2021). Огляд статей з практичного застосування, 15(8), 200-215.

16. Кросс-платформні технології для адаптації пошукових систем на основі вимог роботодавців. (2021). Наукова праця, 5(6), 77-88.

17. Динамічні моделі для покращення пошуку роботи за допомогою штучного інтелекту. (2022). Журнал "Штучний інтелект в економіці", 23(4), 134-145.

18. Моделі прогнозування запитів на ринку праці в Україні. (2022). Науковий журнал, 10(2), 92-103.

19. Рекомендаційні системи для платформ пошуку роботи в Україні. (2021). "Інноваційні рішення у сфері рекрутингу", 9(4), 65-78.

20. Моделювання роботи автоматизованих систем підбору кадрів для підприємств в Україні. (2022). Журнал "Технічні рішення в бізнесі", 19(1), 112-124.

21. Інформаційні технології для поліпшення процесу пошуку роботи за допомогою аналізу даних. (2022). Публікація на сайті Microsoft. https://azure.microsoft.com/

22. Платформи для онлайн-пошуку роботи та вдосконалення алгоритмів підбору кадрів. (2020). Практичні рекомендації для HR спеціалістів. https://www.hr-tech.com/

23. Аналіз та оптимізація процесів пошуку роботи за допомогою технологій Azure. (2021). https://azure.microsoft.com/

24. Документація по використанню Angular. (2023). https://angular.io/docs

25. Моделі машинного навчання для оптимізації пошуку роботи на основі даних користувачів. (2022). https://learn.microsoft.com/uk-ua/azure/machine-learning/

26. Можливості SQL Server для аналітики на ринку праці. (2023). Офіційна документація SQL Server, https://learn.microsoft.com/uk-ua/sql/

27. Документація по Microsoft Power BI для аналізу даних на ринку праці. (2023). https://learn.microsoft.com/uk-ua/power-bi/

28. Azure SQL Database для зберігання та аналізу даних про вакансії та резюме. (2023). https://learn.microsoft.com/uk-ua/azure/azure-sql/

29. AI та автоматизація пошуку роботи на платформі Microsoft Dynamics 365. (2023). https://learn.microsoft.com/uk-ua/dynamics365/

30. Інтеграція систем штучного інтелекту та даних користувачів для пошуку роботи. (2021). Стаття в українському журналі з IT. https://it-journal.com.ua/

31. Використання AI в автоматизованому підборі персоналу в Україні. (2020). Доклад на науковій конференції. https://hrtech.com.ua/

32. Martin, R. C. (2018). "Чиста архітектура: Програмування для професіоналів". https://www.amazon.com/Clean-Architecture-Craftsmans-Software-Structure/dp/0134494164.

33. Udemy. Курс "Implementing Clean Architecture in .NET". (2023). https://www.udemy.com/course/clean-architecture-dotnet/

34. Штучний інтелект в роботі систем підбору персоналу на платформі Dynamics 365. (2021). https://learn.microsoft.com/uk-ua/dynamics365/

35. Створення інтелектуальних платформ для підбору персоналу через великий обсяг даних. (2022). Наукова публікація в Україні, https://it-journal.com.ua/

36. Документація по використанню Microsoft SQL Server для обробки даних у рекрутингу. (2023). https://learn.microsoft.com/uk-ua/sql/

37. Офіційний блог ASP.NET. (2023). https://devblogs.microsoft.com/aspnet/

38. Udemy. Курс "Complete Guide to ASP.NET Core MVC". (2023). https://www.udemy.com/

ДОДАТКИ

Додаток А

Фрагменти програмниих лістингів

```
        namespace Teamnety.Web.Api.Areas.Admin.Controllers.Jobs
        {
            /// <inheritdoc />
            /// <summary>
            /// Jobs API Admin Controller
            /// </summary>
            [Route("api/admin/[controller]")]
            [Authorize(Roles = Consts.SystemAdmin + "," + Consts.CompanyAdmin,
AuthenticationSchemes = IdentityServerAuthenticationDefaults.AuthenticationScheme)]
            public class JobsController : BaseController
            {
                private readonly IMapper _mapper;
                private readonly INotificationService _notificationService;
                private readonly IJobService _jobService;
                private readonly IProfileService _profileService;
                /// <summary>
                /// Jobs Controller Constructor
                /// </summary>
                /// <param name="workContext"></param>
                /// <param name="jobService"></param>
                /// <param name="profileService"></param>
                /// <param name="notificationService"></param>
                /// <param name="mapper"></param>
                public JobsController(IWorkContext workContext,
                    IJobService jobService,
                    IProfileService profileService,
                    INotificationService notificationService,
                    IMapper mapper) : base(workContext)
                {
                    _jobService = jobService;
                    _profileService = profileService;
                    _notificationService = notificationService;
                    _mapper = mapper;
                }

                /// <summary>
                /// The update API allows to update a job item based on provided JSON
object.
                /// According to the HTTP specification, a PUT request requires the
client to send the entire updated entity, not just the deltas.
                /// To support partial updates, use HTTP PATCH.
                /// </summary>
                /// <param name="id">Job id.</param>
                /// <param name="reviewStatus">Review Status.</param>
                /// <returns>The response is 204 (No Content).</returns>
                [HttpPut("{id}")]
                [ProducesResponseType(200)]
                [ProducesResponseType(400)]
                public async Task<IActionResult> UpdateJobStatusAsync(int id,
[FromBody] JobReviewStatus reviewStatus)
                {
                    var job = await _jobService.GetJobById(id);
                    if (job is null)
                        return NotFound();

                    job.ReviewStatus = reviewStatus;
```

```
                    job.IsInActive = reviewStatus == JobReviewStatus.Inactive ? true
                        : reviewStatus == JobReviewStatus.Active ? false
                        : job.IsInActive;

                    job.IsPublished = reviewStatus switch
                    {
                        JobReviewStatus.Active => true,
                        _ => false
                    };

                    _jobService.Update(job);
                    var culture =
_profileService.GetCultureByProfileId(job.CreatedBy.Profile.Id).DefineNotificationLan
guage();

                    if (job.CreatedBy.Email != null)
                        await _jobService.SendEmailsForUser(job, job.CreatedBy.Email,
culture);

                    var mappedJob = _mapper.Map<JobTableItemDto>(job);

                    return Ok(mappedJob);
                }

                /// <summary>
                /// The delete API allows to delete job based on id provided.
                /// </summary>
                /// <param name="id">Job id</param>
                /// <returns>The Delete response is 204 (No Content).</returns>
                [HttpDelete("{id}")]
                [ProducesResponseType(200)]
                [ProducesResponseType(404)]
                public async Task<IActionResult> DeleteAsync(int id)
                {
                    var job = await _jobService.FindAsync(id);
                    if (job is null)
                        return NotFound();

                    job.IsDeleted = true;

                    _jobService.Update(job);

                    return Ok(new { deletedJobId = id });
                }

                /// <summary>
                /// Get all jobs
                /// </summary>
                /// <returns></returns>
                [HttpPost("table")]
                [ProducesResponseType(400)]
                [ProducesResponseType(200)]
                public async Task<IActionResult> GetJobsForTableAsync([FromBody]
JobsTableFilter filter)
                {
                    var jobs = await _jobService.GetJobsForTableAsync(filter);
                    var model = new AdminJobTableViewModel
                    {
                        Jobs = jobs,
                        RecordsFiltered = jobs.Count
                    };

                    return Ok(model);
                }
```

```
            }
        }

        namespace Teamnety.Web.Api.Controllers.Devices
        {
            /// <inheritdoc />
            /// <summary>
            /// Device API Controller
            /// </summary>
            [Authorize(AuthenticationSchemes =
IdentityServerAuthenticationDefaults.AuthenticationScheme)]
            [Route("api/admin/[controller]")]
            [ApiController]
            public class DeviceController : BaseController
            {
                private readonly IDeviceService _deviceService;
                private readonly IDeviceActionService _deviceActionService;
                private readonly IDeviceActionElementService
_deviceActionElementService;
                private readonly UserManager<ApplicationUser> _userManager;
                private readonly IMapper _mapper;

                /// <summary>
                /// Device Controller Constructor
                /// </summary>
                /// <param name="workContext"></param>
                /// <param name="deviceService"></param>
                /// <param name="deviceActionService"></param>
                /// <param name="userManager"></param>
                /// <param name="deviceActionElementService"></param>
                /// <param name="mapper"></param>
                public DeviceController(
                    IWorkContext workContext,
                    IDeviceService deviceService,
                    IDeviceActionService deviceActionService,
                    IDeviceActionElementService deviceActionElementService,
                    UserManager<ApplicationUser> userManager,
                    IMapper mapper) : base(workContext)
                {
                    _deviceService = deviceService;
                    _deviceActionService = deviceActionService;
                    _deviceActionElementService = deviceActionElementService;
                    _userManager = userManager;
                    _mapper = mapper;
                }

                /// <summary>
                ///     The Get API returns a list of elements with their IDs and
names.
                /// </summary>
                /// <returns>
                ///     If no elements are found, the method returns a 404 error.
                ///     Otherwise, the method returns a 200 status with a JSON
response containing the list of elements.
                /// </returns>
                [HttpGet("elements")]
                [ProducesResponseType(200)]
                [ProducesResponseType(404)]
                public async Task<IActionResult> GetElementsAsync()
                {
                    var elements = await
_deviceActionElementService.GetAllElementsAsync();
                    if (elements is null)
                        return NotFound();
```

```
                return Ok(new { elements = elements });
            }

            /// <summary>
            /// Get device
            /// </summary>
            /// <param name="id"></param>
            /// <returns></returns>
            [HttpGet("{id}")]
            [ProducesResponseType(400)]
            [ProducesResponseType(200)]
            public async Task<IActionResult> GetDevice(int id)
            {
                var device = await _deviceService.FindAsync(id);

                var returnedDivece =
_mapper.Map<AdminDeviceItemViewModel>(device);

                return Ok(returnedDivece);
            }

            /// <summary>
            /// Get device analytics
            /// </summary>
            /// <param name="filter"></param>
            /// <returns></returns>
            [HttpPost("devices-analytics")]
            [ProducesResponseType(400)]
            [ProducesResponseType(200)]
            public async Task<IActionResult> GetDevicesAnalytics([FromBody]
DeviceFilter filter)
            {
                var devicesAnalitics = await
_deviceService.DevicesAnalytics(filter);

                return Ok(devicesAnalitics);
            }

            /// <summary>
            /// Get device actions by device id
            /// </summary>
            /// <param name="filter"></param>
            /// <returns></returns>
            [HttpPost("table-device-actions")]
            [ProducesResponseType(400)]
            [ProducesResponseType(200)]
            public async Task<IActionResult>
GetDeviceActionsByDeviceId([FromBody] DeviceActionsFilter filter)
            {
                var deviceActions = await
_deviceActionService.GetAllFilteredDeviceActionsAsync(filter);

                var deviceActionsMap =
_mapper.Map<IList<DeviceActionViewModel>>(deviceActions.Entities);

                var model = new AdminDeviceActionsTableViewModel
                {
                    DeviceActions = deviceActionsMap,
                    RecordsFiltered = deviceActionsMap.Count
                };

                return Ok(model);
            }
```

```
                /// <summary>
                /// Get all devices
                /// </summary>
                /// <returns></returns>
                [HttpPost("table-device")]
                [ProducesResponseType(400)]
                [ProducesResponseType(200)]
                public async Task<IActionResult> GetDevices([FromBody] DeviceFilter
filter)
                {
                    var devices = await
_deviceService.GetAllFilteredDevicesAsync(filter);

                    var returnedDevicesList =
_mapper.Map<IList<AdminDeviceItemViewModel>>(devices.Entities);

                    var model = new AdminDeviceTableViewModel
                    {
                        Devices = returnedDevicesList,
                        RecordsFiltered = returnedDevicesList.Count
                    };

                    return Ok(model);
                }
            }
        }

        namespace Teamnety.Web.Api.Areas.Admin.Controllers.Jobs
        {
            /// <inheritdoc />
            /// <summary>
            /// HotJobHistory API Admin Controller
            /// </summary>
            [Route("api/admin/[controller]")]
            [Authorize(Roles = Consts.SystemAdmin + "," + Consts.CompanyAdmin,
AuthenticationSchemes = IdentityServerAuthenticationDefaults.AuthenticationScheme)]
            public class HotJobHistoryController : BaseController
            {
                private readonly IMapper _mapper;
                private readonly IHotJobHistoryService _hotJobHistoryService;
                private readonly ICompanyService _companyService;

                /// <summary>
                ///     HotJobHistory Controller Constructor
                /// </summary>
                /// <param name="workContext"></param>
                /// <param name="hotJobHistoryService"></param>
                /// <param name="mapper"></param>
                /// <param name="companyService"></param>
                public HotJobHistoryController(IWorkContext workContext,
                    IMapper mapper,
                    IHotJobHistoryService hotJobHistoryService,
                    ICompanyService companyService)
                    : base(workContext)
                {
                    _mapper = mapper;
                    _hotJobHistoryService = hotJobHistoryService;
                    _companyService = companyService;
                }

                /// <summary>
                ///     Create hot job history item
                /// </summary>
```

```
                /// <param name="dto">HotJobHistory view model</param>
                /// <returns>Returns a 200 response.</returns>
                [HttpPost]
                [ProducesResponseType(200)]
                public async Task<IActionResult> Add([FromBody]
HotJobHistoryCreateDto dto)
                {
                    var result = await
_hotJobHistoryService.CreateHotJobCountAsync(dto);
                    if (result is null)
                        return BadRequest();

                    return Ok(result);
                }

                /// <summary>
                ///     The update API allows to update a HotJobHistory item based on
provided JSON object.
                ///     According to the HTTP specification, a PUT request requires
the client to send the entire updated entity, not just
                ///     the deltas.
                ///     To support partial updates, use HTTP PATCH.
                /// </summary>
                /// <param name="dto">Job comment view model.</param>
                /// <returns>The response is 200.</returns>
                [HttpPut]
                [ProducesResponseType(200)]
                [ProducesResponseType(404)]
                public async Task<IActionResult> UpdateAsync([FromBody]
HotJobHistoryDto dto)
                {
                    var result = await
_hotJobHistoryService.UpdateHotJobHistoryAsync(dto);
                    if (result is null)
                        return BadRequest();

                    return Ok(result);
                }

                /// <summary>
                ///     The delete API allows to delete HotJobHistory item based on
id provided.
                /// </summary>
                /// <param name="id">HotJobHistory id</param>
                /// <returns>The Delete response is 200.</returns>
                [HttpDelete]
                [ProducesResponseType(204)]
                [ProducesResponseType(404)]
                public async Task<IActionResult> Delete(int id)
                {
                    var result = await
_hotJobHistoryService.DeleteHotJobHistoryAsync(id);
                    if (result == null)
                        return NotFound();

                    return Ok(result);
                }
            }
        }
```