ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА

до бакалаврської роботи

на ступінь вищої освіти бакалавр

на тему: «**Розробка Web-платформи для агрегації даних сайтів пошуку роботи мовою C#**»

Виконав: студент 4 курсу, групи ПД-41

спеціальності

121 Інженерія програмного забезпечення

(шифр і назва спеціальності)

Лебідь М.М.

(прізвище та ініціали)

Керівник Трінтіна Н.А.

(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Київ–2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти - «Бакалавр»

Напрям підготовки -121-“Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____О.В. Негоденко

“____”_____2023 року

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Лебедю Максиму Максимовичу

(прізвище, ім’я, по батькові)

1. Тема роботи: «Розробка Web-платформи для агрегації даних сайтів пошуку роботи мовою C#»

Керівник роботи Трінтіна Н.А., к.т.н., доц., доцент кафедри ІПЗ

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від “24” лютого 2023 року №26

2. Строк подання студентом роботи «1» червня 2023 року

3. Вихідні дані до роботи:

3.1. Основні положення побудови Web API;

3.2. Методи побудови скраперів ;

3.3. Існуючі агрегатори даних ;

3.4. Науково-технічна література

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

4.1. Аналіз предметної галузі

4.2. Засоби програмної реалізації

4.3. Проектування Web-додатку

4.4. Розробка Web-додатку

4.5. Тестування Web-додатку

4.6. Висновки

5. Перелік Демонстраційного матеріалу (назва основних слайдів)

5.1. Титульний лист

5.2. Мета, об’єкт та предмет дослідження

5.3. Задачі дипломної роботи

5.4. Аналіз аналоів

5.5. Вимоги до програмного забезпечення

5.6. Програмні засоби реалізації

5.7. Діаграма варіантів використання

5.8. Діаграма пакетів

5.9. Схема бази даних

5.10. Екранні форми

5.11 Апробація результатів дослідження

5.12. Висновки

6. Дата видачі завдання «25» лютого 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ З/п	Назва етапів бакалаврської роботи	Срок виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	25.02.2023-27.02.2023	Виконано
2	Дослідження існуючих web-агрегаторів	27.02.2023-01.03.2023	Виконано
3	Проектування архітектури системи	01.03.2023-25.03.2023	Виконано
4	Розробка платформи	25.03.2023-15.04.2023	Виконано
5	Висновки, оформлення роботи	15.04.2023-08.05.2023	Виконано
6	Розробка демонстраційних матеріалів	09.05.2023	Виконано
7	Попередній захист роботи	22.05.2023	Виконано
8	Здача роботи	1.06.2023	

Студент _____ Лебідь М.М.

Керівник роботи_____ Трінтіна Н.А.

РЕФЕРАТ

Текстова частина бакалаврської роботи 55 с. ,13 рис. , 1 табл. , 12 джерел .

Ключові слова: Visual Studio, Web API, postgresql , C#, Docker, агрегатор.

Об'єкт дослідження – процес агрегації даних з популярних сайтів пошуку роботи .

Предмет дослідження : технології розробки Web-додатока для агрегації даних з популярних сайтів пошуку роботи .

Мета роботи – спрощення процесу агрегації даних з популярних сайтів пошуку роботи за рахунок його автоматизації з використанням веб-додатку мовою C#.

Для досягнення мети необхідно пройти наступні етапи:

1. Аналіз аналогів в схожих Web-платформах;
2. Аналіз технічних засобів та вибір відповідних;
3. Розробка вимог до web-платформи на основі аналізу переваг і с недоліків аналогів;
4. Проектування і розробка web-платформи;
5. Тестування;

Галузь використання – є ринок праці. Зокрема, проект може бути корисним для роботодавців, які шукають кваліфікованих працівників, а також для працівників, які шукають відповідні вакансії.

ЗМІСТ

ВСТУП

Актуальність теми. З появою Інтернету та розвитком технологій, багато людей використовують онлайн-ресурси для пошуку роботи. Однак, з великою кількістю сайтів з вакансіями, важко знайти найбільш підходящу роботу, витрачаючи багато часу на перегляд та аналіз інформації.

Таким чином, розробка платформи, яка може агрегувати та аналізувати дані з різних сайтів пошуку роботи, є дуже важливою і актуальною. Платформа може забезпечити більш ефективний та швидкий пошук вакансій, що відповідають вимогам користувача. Крім того, аналіз даних про ринок праці може допомогти роботодавцям та працівникам зрозуміти тенденції на ринку та зробити більш обґрунтовані рішення щодо пошуку та відбору працівників.

Також важливо зазначити, що в контексті пандемії COVID-19, де відбувається зміна у вимогах до працівників та структурі ринку праці, проект такої платформи може стати особливо корисним для тих, хто шукає роботу або намагається знайти нові можливості на ринку праці.

Об'єкт дослідження – процес агрегації даних з популярних сайтів пошуку роботи .

Предмет дослідження : технології розробки Web-додатока для агрегації даних з популярних сайтів пошуку роботи .

Мета роботи – спрощення процесу агрегації даних з популярних сайтів пошуку роботи за рахунок його автоматизації з використанням веб-додатку мовою C#

Для досягнення мети необхідно пройти наступні етапи:

- Аналіз аналогів в схожих Web-платформах;
- Аналіз технічних засобів та вибір відповідних;
- Розробка вимог до web-платформи на основі аналізу переваг і недоліків аналогів;

- Проектування і розробка web-платформи;
- Тестування;

Галузь використання – є ринок праці. Зокрема, проект може бути корисним для роботодавців, які шукають кваліфікованих працівників, а також для працівників, які шукають відповідні вакансії.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1.Огляд галузі ринку праці

У сучасному світі, де ринок праці постійно змінюється і розвивається, сайти пошуку робіт стають все більш актуальними та важливими для роботодавців і працеспроможних осіб. Сайти пошуку робіт є онлайн-платформами, які надають можливість працевлаштованим особам шукати вакансії та взаємодіяти з потенційними роботодавцями. Ці сайти стали невід'ємною частиною сучасного ринку праці, де працівники і роботодавці шукають зручний та ефективний спосіб знаходження відповідних працевлаштувань та кандидатів на роботу. Отримати роботу в Україні стало складніше, що підтверджується даними з сайтів Work.ua та Rabota.ua. За останній рік кількість вакансій зменшилася майже вдвічі, до 52 тисяч і 42,8 тисячі відповідно. Це змушує людей активно шукати роботу, і одним з варіантів є розміщення свого резюме на цих сайтах. За останній рік кількість резюме зросла до 4 мільйонів і 4,7 мільйонів відповідно.

Крім цього, кількість пропозицій від Державного центру зайнятості також зменшилася вдвічі. В січні база центру містила інформацію про 20,5 тисяч вакансій і 94,5 тисячі резюме.

Один з факторів, що призводить до меншої популярності Державного сервісу серед шукачів роботи, полягає в тому, що більшість заявок, розміщених там, походять від державних та комунальних установ із заробітною платою, близькою до мінімально припустимої. [1]

Навіть враховуючи умови які склались на даний момент, кількість вакансій є досить великою, крім цього з часом кількість буде зростати, і як роботадавцям так і працівникам може бути затратно по часу щодня прогортувати різні джерела в пошуку нових вакансій, зручніше було б мати

можливість бачити ці всі дані в одному додатку, саме для цього корисним є застосування агрегаторів.

Агрегатори даних дозволяють користувачам ефективно шукати вакансії, використовуючи розширені функції пошуку, фільтрації та сортування. Крім того, вони можуть надавати додаткові функції, такі як підписка на сповіщення про нові вакансії, збереження улюблених пошукових запитів, створення профілів користувачів тощо.

Загальна мета агрегаторів даних в сфері пошуку роботи полягає в полегшенні процесу пошуку роботи для працевлаштованих осіб шляхом забезпечення зручного та широкого доступу до вакансій з різних джерел. Вони допомагають скоротити час і зусилля, витрачені на пошук вакансій, і сприяють більш ефективному взаємодії між працівниками та роботодавцями.

1.2 Аналіз алгоритмів агрегації даних

Агрегація даних – це процес узагальнення великого обсягу інформації з метою проведення аналізу на високому рівні. На основному рівні цей процес включає збір інформації з різних джерел даних і її організацію в зручний формат, який часто використовує суми, середні значення, медіани або інші статистичні показники. Важливо зазначити, що для агрегації можна використовувати не лише числові дані, а й нечислові елементи, які також можна підрахувати[2].

Для отримання корисних і точних результатів, необхідно пройти кілька основних етапів у процесі агрегування даних.

- ***Колекція:***

Агрегація даних розпочинається зі збору фактичних даних. Інструмент агрегації витягує дані з різних джерел і зберігає їх у великих базах даних у вигляді атомарних значень. Перед агрегацією важливо перевірити точність даних і забезпечити достатню кількість інформації. Наприклад,

прогнозування виборів засноване на результатів голосування, коли доступно лише 10% даних, може привести до неточного прогнозу.

Вибір техніки агрегації:

Існує кілька технік агрегації даних, залежно від обсягу і типу даних. Наприклад:

- Внутрішньомережне агрегування: використовується система з кількома етапами для збору та передачі інформації.
- Деревоподібний підхід: дані порівнюються та агрегуються від листків до кореня дерева.
- Підхід на основі кластерів: використовується для порівняння великих обсягів даних у всій мережі.
- Багатошляховий підхід: частково агреговані дані передаються до кореневого вузла, який далі може відправляти дані по різних шляхах.

Визначення інтервалів збору даних:

Необхідно встановити інтервали, протягом яких збираються дані для агрегації. Наприклад, зведена таблиця може містити дані, зібрані з певного мережевого пристрою протягом тижня. Інтервали збору даних можуть бути щоденними, тижневими, квартальними або річними. Важливо визначити, який період є найбільш відповідним для збору даних, враховуючи потреби і цілі аналізу.

Звітний період:

Звітний період визначає тривалість збору даних. Наприклад, можуть бути створені зведені таблиці з даними, зібраними за певний період, наприклад, за тиждень. Звітний період може включати як необроблені дані, так і агреговані значення. Звітні періоди можуть бути щоденними, тижневими, квартальними або річними, залежно від потреб і контексту аналізу.

Деталізація:

Деталізація визначає період, протягом якого збираються точки даних для конкретного ресурсу або набору ресурсів для агрегації. Наприклад, якщо потрібно обчислити середнє значення точок даних для певного ресурсу за

кожні 10 хвилин, то деталізація становить 10 хвилин. Важливо встановити деталізацію відповідно до потреб аналізу і рівня деталізації, який необхідний для отримання корисних висновків.

Період опитування:

Період опитування визначає часовий інтервал, з яким ресурси використовуються для вибірки даних. Наприклад, може бути встановлено, що дані збираються кожну годину, кожні 15 хвилин або інший визначений проміжок часу. Період опитування впливає на актуальність даних і потребу у відповідності до швидкодіючих або змінних процесів.

Обробка:

Обробка даних включає необхідні кроки для витягування даних і їх подальшої обробки. Цей процес використовує певні інструменти для визначення атомарних даних, які будуть агреговані, та може застосовувати такі процеси, як прогнозний аналіз, алгоритми машинного навчання та штучний інтелект. Для отримання зведених даних застосовуються спеціальні статистичні функції, такі як середні значення, медіани та середні значення.

Презентація:

Після обробки дані представляються узагальненим форматом для подальшого аналізу, розуміння та огляду. Це означає, що зведені дані структуруються та візуалізуються, щоб зробити їх зрозумілими та доступними для використання.

Автоматизоване агрегування проти ручного:

Дискусія про використання автоматизованого або ручного підходу до агрегації даних продовжується. Часто компанії намагаються виконати агрегацію даних вручну, що вимагає значної довіри до людей. Однак ручний процес агрегації може бути складним, інтенсивним та супроводжуватися ризиком помилок. Він також може бути неефективним, неспроможним масштабуватися та призводити до втрати доходу для підприємства.

У сучасному конкурентному середовищі автоматизований підхід до агрегації даних є більш ефективним та надійним. Він допомагає уникнути помилок, звільняє ресурси, прискорює процес та дозволяє підприємствам бути більш гнучкими та інноваційними. Використання автоматизованого підходу до агрегації даних дозволяє ефективно управляти великим обсягом даних і використовувати їх для отримання цінної інформації.

Однак, вибір правильного інструменту для агрегації даних залежить від потреб кожної конкретної організації. Різні компанії можуть мати відмінні вимоги та специфічні потреби. Наприклад, для медичного закладу можуть бути важливі функції прогнозування і прогнозне моделювання, тоді як для ресторану цікавим може бути зручний інтерфейс користувача. Тому перед вибором інструменту важливо провести детальний аналіз потреб і вимог організації. Розмова з експертами з обробки даних, які розуміють як необроблені дані, так і необхідні результати, може допомогти прийняти правильне рішення.

Загалом, автоматизована обробка і агрегація даних є ключовими факторами успіху для підприємств у сучасному світі. Вони допомагають підвищити ефективність, знизити ризики помилок і забезпечити швидкий доступ до цінних даних, що сприяє подальшому розвитку та інноваціям.

Використання агрегованих даних має широкі застосування в різних сферах і галузях. Агрегація даних є необхідною для прийняття точних рішень і сприяє інноваціям у продуктах, плануванні на майбутнє та оптимізації операцій. Ось кілька прикладів використання агрегованих даних:

- Роздрібна торгівля: аналіз конкуренції і витрат допомагає виробникам і роздрібним торговцям приймати стратегічні рішення щодо маркетингу і планування продажів.
- Туризм: туристичні компанії використовують агреговані дані для вивчення змін у туристичних уподобаннях, тарифах і нових ринках для планування та адаптації своїх послуг.

- Маркетинг: агрегація даних з маркетингових кампаній дозволяє глибоко аналізувати ефективність кампаній і розуміти поведінку різних аудиторій.
- Охорона здоров'я: державні заклади охорони здоров'я використовують агреговані дані для моніторингу розвитку хвороб, прогнозування тенденцій і розробки інноваційних рішень в галузі медицини.

Хоча агрегація даних має багато переваг, вона також пов'язана з ризиками, зокрема порушенням конфіденційності даних. Для забезпечення правильної обробки даних і використання їх у відповідності до законодавства, необхідно дотримуватися правил і механізмів захисту приватності.

Загалом, агрегація даних є незамінним інструментом у світі сучасного бізнесу, де точність рішень і вміння адаптуватися до змін є критичними для успіху. Вона дозволяє підприємствам збирати, аналізувати та використовувати інформацію з різних джерел для досягнення своїх цілей і отримання конкурентної переваги.

1.3 Аналіз існуючих рішень

Аналіз існуючих рішень є важливим етапом при розробці нової платформи для пошуку роботи . Цей аналіз допомагає виявити прогалини і можливості, визначити конкурентну перевагу, вивчити кращі практики в галузі та забезпечити унікальність і оригінальність нової платформи. Виявлення переваг та недоліків існуючих рішень допомагає вдосконалити функціонал та якість роботи нової платформи, а також знайти способи надання конкурентної переваги. Використання кращих практик і уникнення повторення помилок існуючих рішень сприяє успіху та задоволенню користувачів.

1.3.1 Огляд наявних веб-платформ для пошуку роботи

Indeed - це популярна платформа пошуку роботи, яка є цінним джерелом інформації як для шукачів роботи, так і для роботодавців. Він об’єднує списки вакансій із різних джерел, включаючи веб-сайти компаній, рекламні дошки та агентства з працевлаштування, надаючи вичерпну базу даних про доступні вакансії. Користувачі можуть шукати роботу за ключовими словами, місцезнаходженням, галуззю та іншими критеріями.

Glassdoor : Цей агрегатор поєднує вакансії зі сторінками компаній, рейтингами роботодавців та відгуками про робочі місця. Glassdoor дозволяє користувачам отримувати докладну інформацію про компанії та вакансії, а також про заробітну плату та умови праці.

Єдиний портал вакансій- це інноваційна онлайн-платформа, що має на меті забезпечити централізований доступ до всіх вакансій на ринку праці. Цей портал об'єднує різні джерела вакансій, включаючи кар'єрні портали, сайти компаній, державні служби зайнятості та рекрутингові агентства.

Для урядових органів і служб зайнятості єдиний портал вакансій може стати ефективним інструментом для моніторингу ринку праці, аналізу тенденцій та планування соціально-економічних програм. Він надає централізовану базу даних щодо вакансій, яка може бути використана для статистичного аналізу та прийняття рішень у галузі зайнятості.

1.3.2 Переваги та недоліки існуючих рішень

По-перше, Indeed має велику базу вакансій зі всього світу. Ви можете знайти вакансії у різних галузях, від маленьких місцевих компаній до великих міжнародних корпорацій. Така широка географічна присутність дозволяє знайти роботу як у своєму регіоні, так і за кордоном, що є великим плюсом для тих, хто шукає роботу у різних місцях.

Незважаючи на свої переваги, Indeed має деякі недоліки. Перш за все, через велику кількість вакансій може виникати проблема з фільтрацією та

точністю результатів. Іноді Indeed може відображати вакансії, які не повністю відповідають вимогам користувача. Тому важливо уважно переглядати деталі вакансій та переконатися, що вони відповідають вашим очікуванням.

Однією з головних переваг платформи Glassdoor є доступ до реальних відгуків про компанії від їхніх співробітників та колишніх працівників. Користувачі можуть прочитати відгуки про робочі умови, заробітну плату, корпоративну культуру, керівництво та багато іншого. Ця інформація дозволяє потенційним працівникам отримати більш об'єктивне уявлення про компанію та зробити більш об’єднане рішення про свої трудові перспективи.

Одним з недоліків Glassdoor є можлива необ'єктивність та суб'єктивність деяких відгуків. Платформа дає можливість користувачам анонімно залишати відгуки, і це може призводити до ситуацій, коли деякі відгуки можуть бути перекрученими або неправдивими. Без можливості перевірити автентичність відгуків, користувачам важко розрізнити об'єктивну інформацію від особистих думок або неправдивих повідомлень. Потрібно здійснювати критичне мислення та перевіряти інформацію з інших джерел, щоб отримати більш повну картину про компанію.

Сервіс дозволяє користувачам знайти інформацію про роботу шляхом пошуку за назвою, регіоном, галуззю та очікуваним рівнем зарплати. Коли користувач здійснює запит, сервіс відображає список доступних вакансій, що відповідають його вимогам, разом з описом робочих вимог та умов для потенційного співробітника. Якщо користувач зацікавлений у певній роботі, він може надіслати своє резюме та відгукнутися на пропозицію, перейшовши на платформу, де роботодавець розмістив дану вакансію. Але не зважаючи на варіативність вибору і ряд додаткових фільтрів цей портал може мати певні недоліки щодо валідності вакансій, а саме:

- вакансія може бути не актуальною
- вакансія може швидко порушувати вимоги українського законодавства(спам/шахрайство і тд.)

- недійсні контакти

Таблиця 1.1. Порівняння аналогів та виявлення серед них схожих ознак

Показник	**Indeed**	**Glassdoor**	**Єдиний портал вакансій**	***JobAsistant***
Велика кількість вакансій:	+	-	+	+
Покриття географії	+	-	+	+
Фільтрація та розширені пошукові функції	+	+	-	+
Інформація про компанії:	-	+	-	-
Перегляд статистики по вакансіям за проміжки часу	-	-	-	+
Сповіщання користувачів по цікавим для них позиціях при додаванні вакансій для них	-	-	-	+
Платні пакети та підписки:	+	+	-	-
Спеціалізація	-	+	-	-

на певних індустріях				
Співпраця з роботодавцями	-	-	-	+

1.3.3 Виявлення прогалин і можливостей для розробки нової платформи

На основі проаналізованих даних про аналоги даного продукту було виявлено що в жодного з розглянутих аналогів не звертається увагу на сповіщення як користувачів так і роботодавців про підходящі їм оголошення. Тому розробка нової платформи з основним акцентом на сповіщення користувачів про цікаві для них позиції та інформування роботодавців про нових потенційних працівників відкриває широкі можливості для поліпшення взаємодії між обома групами користувачів, та надання варіативності щодо розміщення оголошень.

Однією з основних можливостей цієї платформи є забезпечення персоналізованих сповіщень для користувачів щодо нових вакансій, що відповідають їхнім інтересам та професійним навичкам. Це дозволяє користувачам бути в курсі актуальних пропозицій та забезпечує їм перевагу при пошуку роботи.

Крім того, платформа може надавати можливість користувачам налаштовувати параметри сповіщень, такі як розташування, галузь, рівень заробітної плати тощо, що дозволяє їм точніше визначати свої вподобання та отримувати релевантну інформацію.

З точки зору роботодавців, платформа надає можливість швидко та ефективно отримувати інформацію про нових потенційних працівників, які відповідають їх вимогам. Роботодавці можуть отримувати повідомлення про

кандидатів, які підходять до їх вакансій, та мати можливість безпосередньо взаємодіяти з ними через платформу.

Крім того, платформа може надати функціонал для аналізу даних та статистики, що дозволяє роботодавцям оцінювати ефективність своїх пропозицій роботи та вдосконалювати свою стратегію підбору персоналу.

Загалом, розробка такої платформи з фокусом на сповіщення користувачів про цікаві вакансії та інформування роботодавців про потенційних працівників має значний потенціал для покращення ефективності процесу пошуку роботи та підбору персоналу. Основні можливості такої платформи включають:

- Персоналізовані сповіщення: Платформа може надавати користувачам можливість налаштувати свої вподобання та отримувати сповіщення про нові вакансії, які відповідають їхнім інтересам та навичкам. Це дозволяє користувачам бути в курсі актуальних пропозицій та збільшує їх шанси на успішне працевлаштування.
- Розширений пошук: Платформа може забезпечувати розширені можливості пошуку, включаючи фільтрацію за різними параметрами, такими як розташування, галузь, рівень заробітної плати та інші. Це допомагає користувачам знаходити більш точні та відповідні вакансії.

РОЗДІЛ 2. ВИБІР ЗАСОБІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

2.1. Мова C#

C# є сучасною об'єктно-орієнтованою мовою програмування, яка працює в середовищі .NET . Вона надає розробникам засоби для створення безпечних та надійних програм. Орієнтована на об'єкти, C# має мовні конструкції, що підтримують цю концепцію, дозволяючи створювати та використовувати програмні компоненти [10].

Особливості мови C# роблять її дуже зручною для використання. Наприклад:

- Складання сміття автоматично вивільняє пам'ять, зайняту об'єктами, які стали недосяжними та не використовуються.
- Типи, які допускають значення null, забезпечують захист від змінних, що посилаються на об'єкти, що не виділені.
- Обробка винятків надає структурований та розширюваний підхід до виявлення помилок та відновлення після них.
- Лямбда-вирази підтримують прийоми функціонального програмування.
- Синтаксис LINQ створює загальний шаблон для роботи з даними будь-якого джерела.
- Підтримка асинхронних операцій забезпечує синтаксис для створення розподілених систем.
- C# має єдину систему типів, де всі типи успадковують від кореневого типу "object". Всі типи використовують спільний набір операцій, і значення будь-якого типу можна зберігати, передавати та обробляти таким чином. Крім того, C# підтримує як користувацькі посилання, так і значення.

- C# дозволяє динамічно виділяти об'єкти та зберігати спрощені структури у стеку.
- C# підтримує універсальні методи та типи, які забезпечують підвищену безпеку типів та продуктивність.
- C# надає можливість використовувати ітератори, які дозволяють розробникам класів колекцій визначати специфічні варіанти поведінки для клієнтського коду.

Всі ці особливості мови C# роблять її потужним і гнучким інструментом для розробки програмного забезпечення. Вона надає розробникам широкі можливості для створення безпечних, ефективних та гнучких програм з використанням об'єктно-орієнтованого підходу та сучасних концепцій програмування.

2.2 ASP.NET Core Web App (MVC)

ASP.NET Core є переробленою та покращеною версією попереднього ASP.NET фреймворку. Вона має переваги, такі як висока продуктивність, переносимість між платформами (Windows, Linux, macOS), легкість розгортання та гнучкість вибору інструментів.

За допомогою ASP.NET Core Web App (MVC), розробники можуть швидко створювати веб-додатки, які мають логічну розділеність між моделлю (Model), представленням (View) та контролером (Controller). Це дозволяє використовувати принцип розділення відповідальностей, що полегшує розробку, тестування та підтримку програмного забезпечення.

Модель представляє дані та бізнес-логіку додатку. Вона відповідає за доступ до даних та їх обробку. Представлення відображає дані користувачу у вигляді веб-сторінок або інших інтерфейсів. Контролер обробляє вхідні запити від користувача, взаємодіє з моделлю для отримання необхідних даних та передає їх до представлення для відображення результату користувачу.

ASP.NET Core Web App (MVC) надає широкий спектр функцій, таких як маршрутизація URL, керування станом сеансу, аутентифікація та авторизація користувачів, кешування, міжнародна локалізація та багато іншого. Вона також підтримує використання різних фронтенд-технологій, таких як HTML, CSS, JavaScript та фреймворки JavaScript, такі як Angular, React або Vue.js. Це дає розробникам можливість створювати багатофункціональні та інтерактивні веб-додатки[3].

ASP.NET Core Web App (MVC) також підтримує розробку API, що дозволяє створювати веб-служби для обміну даними між різними додатками та платформами. Це забезпечує легку інтеграцію з іншими додатками та системами.

Один з ключових аспектів ASP.NET Core Web App (MVC) - це безпека. Він надає вбудовану підтримку для захисту веб-додатків від атак, таких як міжсайтовий скриптінг (XSS), подделка міжсайтових запитів (CSRF), внедрення SQL-запитів та інші. Розробники можуть використовувати різні механізми аутентифікації та авторизації, такі як інтеграцію зі сторонніми постачальниками, токени доступу та ролеву засновану авторизацію, щоб забезпечити безпеку своїх додатків.

Окрім того, ASP.NET Core Web App (MVC) підтримує гнучку конфігурацію, що дозволяє змінювати поведінку додатків залежно від потреб розгортання. Розробники можуть налаштовувати параметри додатків, такі як підключення до бази даних, налаштування середовища, рівень журналювання тощо, без необхідності зміни вихідного коду.

Загалом, ASP.NET Core Web App (MVC) є потужним фреймворком для розробки веб-додатків, який поєднує високу продуктивність, масштабованість, безпеку та гнучкість. Він надає розробникам багато інструментів та можливостей для швидкої розробки та розг ортання веб-додатків різного рівня складності. З використанням шаблону Model-View-Controller (MVC), розробники можуть ефективно організувати свій код,

розділяючи логіку додатку на окремі компоненти, що сприяє чистоті, розширюваності та тестованості проекту.

Основні переваги ASP.NET Core Web App (MVC) включають:

- Висока продуктивність: Фреймворк працює швидко та ефективно завдяки оптимізаціям, підтримці асинхронного програмування та удосконаленому управлінню ресурсами.
- Кросплатформеність: Додатки можуть бути розгорнуті на різних операційних системах, таких як Windows, Linux або macOS.
- Легкість розгортання: Фреймворк надає можливість розгортати додатки у контейнерах, на хмарних платформах або на локальних серверах з використанням простих інструментів.
- Гнучкість: Розробники можуть використовувати різні режими роботи, такі як режим розробки, виробництва або тестування, залежно від потреб проекту.
- Інтеграція з фронтендом: Фреймворк добре поєднується з фронтенд-технологіями та фреймворками, що дозволяє створювати сучасні та інтерактивні веб-інтерфейси.
- Безпека: ASP.NET Core Web App (MVC) має вбудовані засоби захисту від різних видів атак та надає механізми аутентифікації, авторизації та керування доступом.
- Масштабованість: Фреймворк дозволяє розгортати додатки у розподіленому середовищі та швидко масштабувати їх для обробки великого обсягу запитів.

2.3 PostgreSQL

PostgreSQL[4] — це передова система керування реляційними базами даних (RDBMS) із відкритим кодом, відома своєю надійністю, масштабованістю та широким набором функцій. Він пропонує широкий спектр можливостей, що робить його популярним вибором для різних додатків, від невеликих проектів до великомасштабних корпоративних систем.

Відкрите джерело: PostgreSQL є системою управління базами даних з відкритим кодом, що означає, що ви маєте повний доступ до вихідного коду та можливість вносити зміни або пристосовувати його до потреб вашого проекту. Це надає більшу гнучкість та контроль над базою даних.

Надійність та стабільність: PostgreSQL відомий своєю надійністю та стабільністю. Він має вбудовану механізм контролю цілісності даних, механізми відновлення після відмови та резервне копіювання, що робить його оптимальним вибором для веб-додатків, де надійність та безпека даних є критичними.

Підтримка розширень: PostgreSQL надає широкі можливості для розширення функціональності бази даних за допомогою розширень та налаштувань. Ви можете використовувати розширення, які відповідають потребам вашого проекту, такі як повнотекстовий пошук, географічні функції, робота з JSON-даними тощо.

Підтримка ACID: PostgreSQL гарантує дотримання властивостей ACID (атомарність, консистентність, ізольованість, довершеність) для транзакцій. Це забезпечує цілісність та надійність ваших даних під час операцій зміни даних.

Розширена функціональність: PostgreSQL має багатий набір вбудованих функцій та можливостей. Він підтримує різні типи даних, включаючи рядкові, числові, дати, час

PostgreSQL є однією з найбільш надійних та стабільних баз даних, що забезпечує безперебійну роботу вашого веб-додатку. Він має довгий шлях успішної експлуатації та широке співтовариство розробників, що підтримує його функціональність та безпеку.

Гнучкість та розширені можливості: PostgreSQL надає широкий набір функцій та можливостей для роботи з даними. Він підтримує різноманітні типи даних, включаючи географічні дані, масиви, JSON та багато іншого. База даних також підтримує складні запити, агрегаційні функції, тригери та збережені процедури, що дозволяє зручно та ефективно обробляти дані.

Масштабованість: PostgreSQL має розширені можливості масштабування. Ви можете розподіляти дані на різні сервери або використовувати реплікацію, щоб забезпечити високу доступність та швидкодію вашого додатка. Це дає можливість гнучко налаштовувати базу даних під потреби вашого проекту.

Сумісність зі стандартами: PostgreSQL дотримується стандартів SQL та має високий рівень сумісності з іншими базами даних. Це означає, що ви можете легко переносити або імпортувати дані з інших систем управління базами даних, спрощуючи процес роботи з існуючими даними.

2.4 Вибір додаткових бібліотек

2.4.1.NLog

Логування в проекті є важливим інструментом, який дозволяє фіксувати події та помилки, що виникають під час виконання програмного коду. Це забезпечує можливість відстежувати та аналізувати роботу програми, виявляти та виправляти помилки, поліпшувати якість та надійність проекту. Логи також допомагають в процесі відладки, спрощуючи виявлення та виправлення проблем. Крім того, вони можуть служити джерелом цінної

інформації для аналізу продуктивності, виявлення проблем з безпекою та вдосконалення функціональності проекту.

NLog[6] - це потужна бібліотека журналювання (logging) для платформи .NET, яка може бути оптимальним вибором для вашого проекту. Загалом бібліотека має ряд переваг для проекту а саме:

- Гнучкість налаштування: NLog надає широкі можливості налаштування, що дозволяє пристосувати його до ваших потреб. Ви можете встановлювати рівні журналювання, вибирати різні цільові медіа, форматувати повідомлення та розробляти власні розширення.
- Висока продуктивність: NLog розроблений з урахуванням швидкодії, що дозволяє мінімізувати вплив на продуктивність вашого додатку. Він оптимізований для ефективного журналювання без затримок чи накладних витрат.
- Розширені можливості фільтрації: NLog дозволяє налаштовувати рівні журналювання для різних компонентів вашої програми. Ви можете задавати правила фільтрації, що дозволяють обмежувати обсяг журнальної інформації до потрібного рівня деталей.
- Різноманітні цільові медіа: NLog підтримує запис журнальних повідомлень в різні цільові медіа, такі як файли, бази даних, консоль, сетеві протоколи тощо. Це дає вам свободу вибору зручного способу зберігання та аналізу журнальних даних.
- Інтеграція з іншими компонентами: NLog добре інтегрується з іншими популярними бібліотеками та фреймворками .NET, що дозволяє легко поєднувати його з іншими інструментами, які можна використати у проекті.

2.4.2. Quartz.NET

Quartz.NET[7] - це популярна відкрита бібліотека для планування завдань і планувальників в середовищі .NET. Вона надає розширені

можливості для автоматизації виконання різних задач, таких як запуск фонових процесів, планування повторюваних завдань, планування та виконання пакетних операцій тощо.

Quartz.NET дозволяє визначати розклади для виконання завдань за заданими правилами та з урахуванням різних критеріїв, таких як часові інтервали, часові зони, календарні виключення тощо. Вона також надає можливості керування планувальником, динамічно додавати та видаляти завдання, змінювати їх параметри та стан.

Quartz.NET підтримує різні типи тригерів, такі як тригери на основі календаря, тригери на основі часових інтервалів, тригери на основі подій, що дозволяють гнучко налаштовувати виконання завдань.

Ця бібліотека є потужним інструментом для автоматизації завдань у проектах на платформі .NET, і вона широко використовується в різних сферах, таких як розробка веб-додатків, обробка даних, системи моніторингу

2.5 Visual Studio

Visual Studio[5] - це інтегроване середовище розробки (IDE), розроблене компанією Microsoft, яке надає зручні та потужні інструменти для розробки різноманітних програмних проектів. Воно підтримує багато мов програмування, включаючи C#, C++, Visual Basic, Python та інші.

Visual Studio надає розширені функції редагування коду, такі як підсвічування синтаксису, автодоповнення та інструменти для рефакторингу. Це сприяє підвищенню продуктивності розробника та полегшує написання чистого та ефективного коду.

IDE має вбудовані засоби для налагодження програм, що допомагають виявляти та виправляти помилки. Воно надає можливість встановлювати точки зупинки, крокувати по коду, аналізувати змінні та стежити за виконанням програми.

Незважаючи на свою потужність, Visual Studio є досить простим у використанні і доступним для розробників будь-якого рівня. Інтерфейс IDE є інтуїтивно зрозумілим, з підсказками та допомогою, які допомагають новачкам швидко освоїти його. Крім того, Visual Studio постійно оновлюється і випускає нові версії з поліпшеннями та новими функціями, що робить його ще більш ефективним та зручним для розробки програмного забезпечення.

Visual Studio також має широкий вибір розширень та плагінів, які дозволяють налаштувати середовище розробки згідно з потребами проекту та розширити його функціональність.

Загалом, Visual Studio є потужним та універсальним інструментом для розробки програмного забезпечення, яке допомагає розробникам зосередитися на творчому процесі та підвищує продуктивність роботи над проектами.

РОЗДІЛ 3 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Моделювання об’єкту проектування

2.1.1. Діаграма бази даних

ER діаграми(ERD) означає Entity-Relationship Diagram. Це візуальне представлення сутностей (об’єктів або понять) у системі бази даних і зв’язків між ними. ERD допомагає проілюструвати логічну структуру бази даних і зазвичай використовується на етапах проектування та розробки систем баз даних.

У ERD сутності представлені у вигляді прямокутників, зв’язки – у вигляді ромбів, а атрибути (властивості чи характеристики) – у вигляді овалів або еліпсів. Сутності пов’язані зв’язками, які вказують, як сутності пов’язані одна з одною.

Основні компоненти ERD включають:

- Сутності: це об’єкти або поняття в системі бази даних. Прикладами сутностей можуть бути «клієнт», «продукт» або «працівник».
- Відносини: вони представляють асоціації або зв’язки між сутностями. Відносини можуть бути «один до одного», «один до багатьох» або «багато до багатьох». Наприклад, клієнт може мати багато замовлень, що представляє зв’язок «один до багатьох» між сутностями «клієнт» і «замовлення».
- Атрибути: це властивості або характеристики сутності. Кожна сутність матиме власний набір атрибутів. Наприклад, сутність «клієнт» може мати такі атрибути, як «customer_id», «name» і «email».
- Кардинальність і множинність: кількість екземплярів однієї сутності може бути пов’язана з іншою сутністю. Множинність представляє мінімальну та максимальну кількість екземплярів однієї сутності, які можуть бути пов’язані з іншою сутністю.

- Використовуючи ERD, ви можете візуально представити структуру вашої системи бази даних, включаючи сутності, зв’язки та атрибути. Це допомагає зрозуміти модель даних і сприяє ефективному спілкуванню між зацікавленими сторонами, дизайнерами та розробниками під час процесу проектування бази даних.
- Первинні ключі: первинний ключ унікально ідентифікує кожен екземпляр сутності. Він представлений у ERD шляхом підкреслення атрибута або атрибутів, які складають первинний ключ.
- Зовнішні ключі: Зовнішній ключ є посиланням на первинний ключ іншої сутності. Він встановлює зв’язок між двома сутностями. У ERD зовнішній ключ представлено як атрибут і зазвичай зображується пунктирним підкресленням.
- Обмеження кардинальності: обмеження кардинальності визначають мінімальну та максимальну кількість входжень однієї сутності, яка може бути пов’язана з іншою сутністю. Загальні символи, які використовуються для представлення обмежень потужності, включають «1» для одного входження, «0..1» для нуля або одного входження та «*» для нуля або більше входжень.
- Обмеження участі: обмеження участі вказують, чи є участь суб’єкта у відносинах обов’язковою (повна участь) чи необов’язковою (часткова участь). Загальна участь представлена подвійною лінією, що з’єднує сутність із відносинами, тоді як часткова участь представлена однією лінією.
- Слабкі сутності: існування слабкої сутності залежить від іншої сутності. Він не має власного унікального ідентифікатора, але покладається на первинний ключ сильної сутності разом із власним частковим ключем для формування складеного ключа. Слабкі сутності представлені в ERD подвійним прямокутником.

- Типи атрибутів. Атрибути можна класифікувати як прості атрибути (наприклад, ім’я, вік) або складені атрибути (наприклад, адреса з підатрибутами, як-от вулиця, місто та поштовий індекс). Вони також можуть бути однозначними або багатозначними атрибутами.

Потрібно памятати , що ERD[8] є високорівневим представленням структури бази даних і не містить таких деталей, як типи даних, обмеження чи індекси. Він в основному використовується для фіксації концептуального та логічного дизайну бази даних перед її перетворенням у фізичну схему.

Під час проектування ERD важливо розуміти вимоги до системи бази даних, ідентифікувати сутності, зв’язки та атрибути, а також визначати їхні властивості та обмеження. Це допоможе створити чітке та вичерпне представлення структури бази даних.

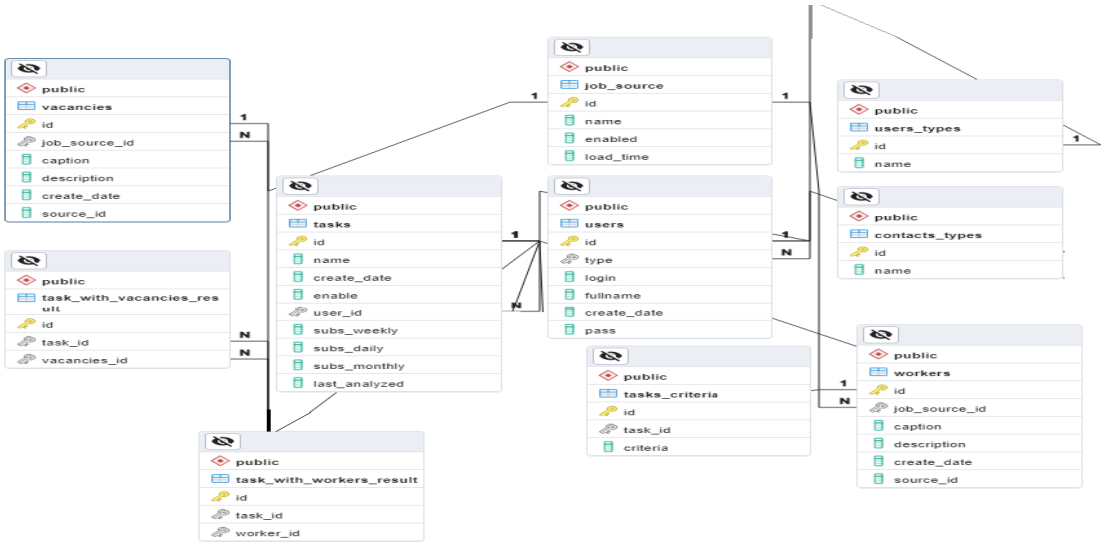


Рисунок 3.1. СХЕМА БД

З діаграми видно , що було задіяно 10 таблиць які будуть використані під час роботи сервісів. Загалом можна виділити що вся інформація яка буде агрегована буде розподілятись між таблицями, і відповідно від потреби буде використовуватись у сервісах , і згідно запитів буде аналізована і виведена користувачу.

3.1.2. Діаграма пакетів

Діаграма пакетів[9] - це структурна діаграма, що ілюструє організацію пакетів або модулів у системі програмного забезпечення. Вона відображає зв'язки та залежності між пакетами, допомагаючи уявити архітектурну структуру системи.

Опис діаграми пакетів включає наступні аспекти:

- Пакети: Визначення основних пакетів або модулів системи, які представляють логічно згруповані колекції класів, інтерфейсів або підсистем.
- Залежності: Вказівка залежностей між пакетами, таких як залежності компіляції або використання.
- Інтерфейси та класи: Відображення публічних інтерфейсів або класів, які доступні в кожному пакеті.
- Модульність: Показ поділу системи на окремі модулі або пакети, що сприяє зручності розробки, тестування та підтримки системи.
- Рівні абстракції: Відображення ієрархії пакетів або рівнів абстракції, де вищі рівні можуть залежати від нижчих, але не навпаки.
- Іменування: Вказівка назв пакетів та модулів для кращого розуміння їх призначення та функціональності.

Опис діаграми пакетів може бути доповнений деталями про залежності між пакетами, розподілом функціональності та відповідальностей між модулями, а також рівнями доступу до пакетів.

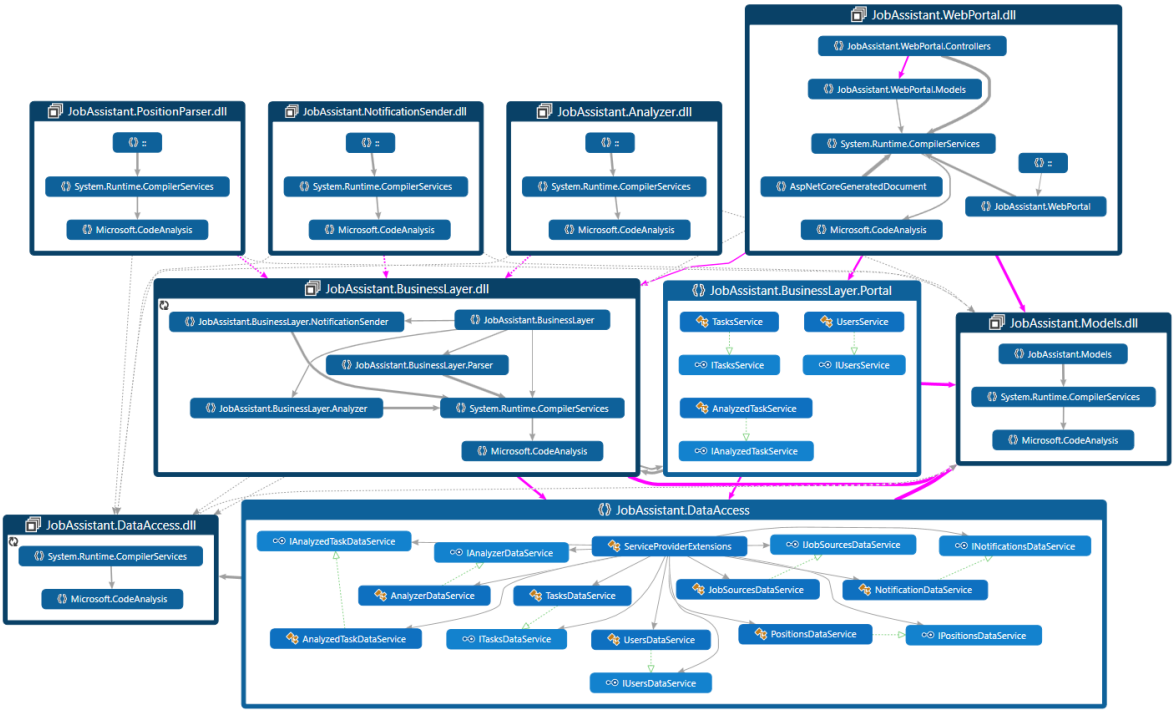


Рисунок 3.2. Діаграма пакетів

3.1.3.Діаграма використання платформи

Діаграма використання додатку[10] (Use Case Diagram) є графічним інструментом уніфікованої мови моделювання (UML), що використовується для візуалізації взаємодій між акторами (користувачами або зовнішніми системами) та системою в рамках певних сценаріїв або функціональних вимог.

Діаграма використання додатку дозволяє встановити основні функціональні можливості системи та показати, як актори взаємодіють з системою для досягнення конкретних цілей. Актори зображаються у вигляді піктограм людей, ролей або зовнішніх систем, а взаємодії відображаються за допомогою стрілок, що показують напрямок комунікації між акторами та системою.

У контексті даної дипломної роботи, було створено діаграму використання додатку для зобразення сценаріїв взаємодій з користувачем, який грає в гру. Інші актори не передбачені, оскільки гра не має багатокористувацького режиму або не взаємодіє з базою даних.

Для створення діаграми були описані всі можливі варіанти використання

системи, тобто сценарії, що описують послідовність кроків та взаємодій між користувачем та системою.

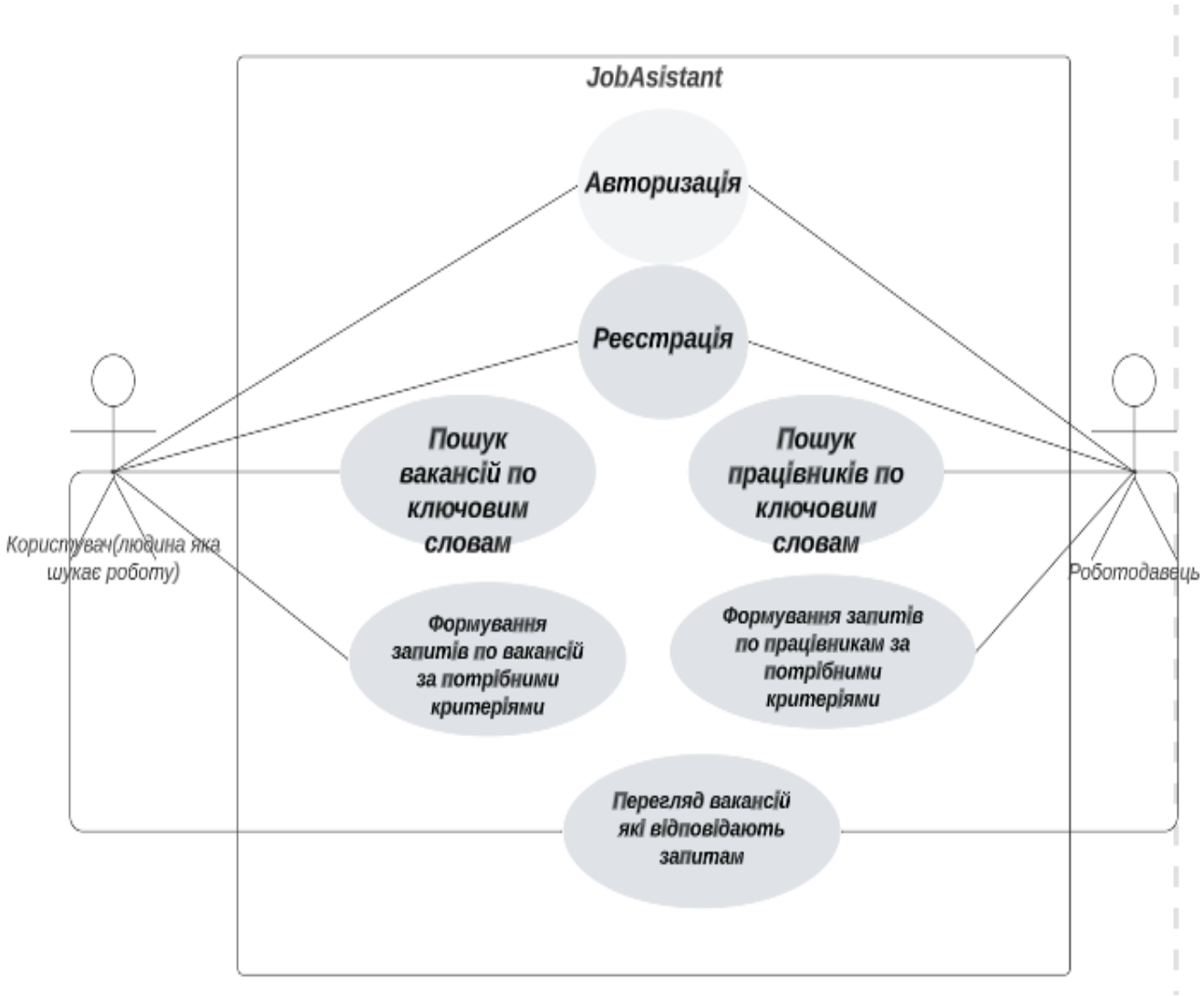


Рисунок 3.3. Діаграма використання

РОЗДІЛ 4. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Планування розробки проекту

На початковому етапі розробки будь-якого програмного забезпечення необхідно визначити поставлені завдання, які потрібно вирішити. Це важливо для того, щоб усі члени розробницької команди мали чітке розуміння того, які завдання треба виконати і які вже були виконані. Для цього я використовував інструмент Trello.

Trello є веб-сайтом та програмою, які допомагають керувати та спрощувати процес створення програмного забезпечення. Цей сервіс є популярним як серед великих компаній, так і серед невеликих стартапів. Він дозволяє ефективно організувати роботу над програмними проєктами за допомогою канбан-дошок, що сприяє зручному відстеженню та керуванню завданнями.

Trello має кілька переваг, серед яких:

- Простота використання:
- Канбан-дошки: Ви можете створювати колонки для різних стадій роботи (наприклад, "To Do", "In Progress", "Done") та переміщати завдання між ними шляхом перетягування. Це дозволяє легко відстежувати прогрес проєкту і управляти завданнями.
- Поєднуваність з іншими схожими програмами
- Наявність мобільні додатка для iOS та Android
- Наявність історії змін Для створення та подальшої реалізації завдань створюються дошки з картками, які можна розділяти в залежності від задач, що в свою чергу дозволяє слідкувати за всіма аспектами створення.

При організації завдань для створення Web додатку було використано стандартні типи дошок а також було додано окрему дошку яка відповідає за

задачі які будуть розроблятися в майбутніх версіях додатку. Дошку відображено на рисунку 4.1.

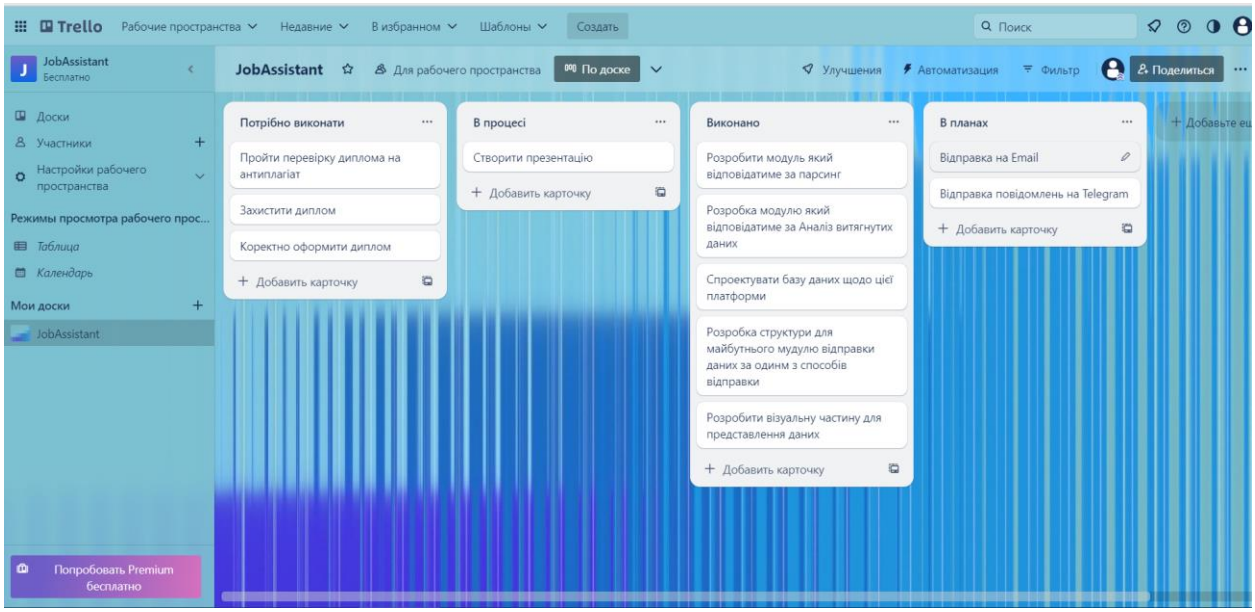


Рис. 4.1. Задачі для створення проєкту на сайті Trello

4.2 Методологія розробки

4.2.1 Agile[11]

Agile - це підхід до розробки програмного забезпечення, який спрямований на гнучкість, співпрацю та постійну адаптацію. У методології Agile розробка програмного забезпечення поділяється на невеликі ітерації, які називаються спринтами. Команда розробників працює над конкретними завданнями протягом кожного спринту, з фокусом на виробництві функціонального та високоякісного продукту.

Гнучкі методології проекту є доволі популярними у наш час. Ці методології розробки дозволяють адаптуватись до вимог, що ставляться перед проектом, та забезпечують ефективне виконання проекту.

4.2.2. Kanban[12]

Хоча гнучкі методології ґрунтуються на спільних принципах, Agile є загальним терміном, який охоплює ці методології, в той час як вони розподіляються на різні підвиди. Так як програмне забезпечення розробляється невеликою командою, було вирішено скористатись методологією Kanban – один із підтипів Agile.

Kanban - це візуальна методологія управління проектами, яка покладає акцент на прозорість, гнучкість та ефективність. Вона виникла в виробничій сфері, але широко використовується в різних галузях, включаючи розробку програмного забезпечення.

Основна концепція Kanban полягає у використанні Канбан-дошки, що є візуальним представленням робочого процесу, що максимально підходить для даного проєкту так як при плануванні використовувалась така дошка. Дошка складається з колонок, які відображають різні етапи проекту або процесу. Кожна колонка містить картки (Канбан-картки), які представляють робочі завдання або задачі. Картки переміщуються по колонках від початку до завершення, що дозволяє візуально відстежувати прогрес та поток роботи.

Основні принципи Kanban включають обмеження робочого навантаження (WIP - Work in Progress), тобто обмеження кількості одночасних завдань, фокус на завершенні поточних завдань перед початком нових, постійне вдосконалення процесу роботи і здійснення змін на основі спостережень та обратного зв'язку.

Канбан дозволяє команді чітко визначити пріоритети, відстежувати прогрес, виявляти перешкоди та швидко реагувати на зміни. Ця методологія сприяє збільшенню продуктивності, зменшенню витрат часу та оптимізації робочого процесу.

4.3. Опис розробки функціонування додатку

4.3.1. Реалізація модуля парсингу даних

Перед початком розробки цього модуля, було проаналізовано обовязкові умови для його, а саме те що цей модуль повинен автоматично планувати і виконувати завдання, тобто самостійно вмикатись раз в певний час і витягувати дані з сайтів пошуку робіт.

Спочатку було створено бібліотеку класів яка має назву JobAssistant.PositionParser, після чого імпортуються необхідні залежності, включаючи класи з різних бібліотек. Далі виконується налаштування логера для ведення журналу подій.

Для виконання цього завдання було налаштоване середовище для виконання додатка, який використовує бібліотеку Quartz Scheduler.

Створюється об'єкт builder типу IHostBuilder, який буде використовуватися для налаштування хоста додатка.

У методі CreateHostBuilder налаштовуються служби, зокрема Quartz Scheduler. Встановлюється ідентифікатор планувальника, використовується фабрика залежностей Microsoft для створення робочих завдань, встановлюється типовий завантажувач, використовується пам'яті зберігання, налаштовується пул потоків та планується виконання ParserSchedulerJob за заданим графіком.

Далі додаються інші служби, пов'язані з Quartz Scheduler, а саме, налаштовується QuartzHostedService, який забезпечує гнучку інтеграцію Quartz Scheduler з хостом додатка.

Нарешті, метод Build() викликається для побудови хоста, а метод Run() запускає виконання додатка. Сам клас **ParserSchedulerJob**, який реалізує інтерфейс **IJob** з бібліотеки Quartz.

Клас **ParserSchedulerJob** має конструктор, який приймає інтерфейси **IJobSourcesDataService**, **IPositionsDataService** та

ILogger<ParserSchedulerJob>. Ці залежності використовуються для отримання даних про джерела робочих місць, зберігання позицій та реєстрації подій.

Метод **Execute** є обов'язковим методом, який буде викликаний при виконанні робочого завдання. У цьому методі виконується наступне:

1. Створюється об'єкт **Stopwatch** для вимірювання часу виконання.
2. Записується інформаційне повідомлення в журналі.
3. Отримуються джерела робочих місць.
4. Створюються порожні списки **parsedVacancies** і **parsedWorkers**.
5. Для кожного джерела робочих місць, яке має включену прапорець **Enabled**, виконується наступне:

- В залежності від назви джерела, створюється відповідний парсер (**JobUaParser**, **HhUaParser**, тощо).
- Вакансії та робітники, отримані з парсера, додаються до відповідних списків.

6. Записуються налагоджувальні повідомлення у журналі, що вказують на кількість вакансій та робітників, які були спарсені.
7. Зберігаються вакансії та робітники за допомогою **IPositionsDataService**.
8. Записується налагоджувальне повідомлення у журналі, яке показує загальний час виконання.

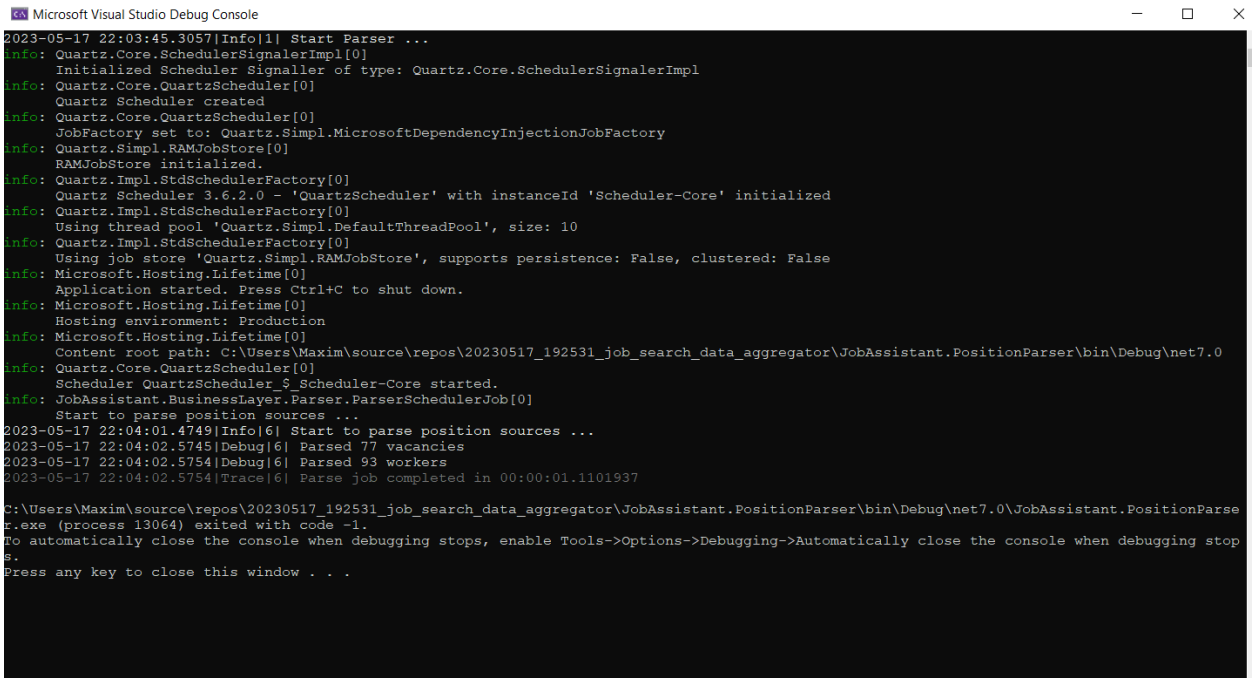


Рисунок 4.2. Відповідні повідомлення в журналі **parser**

Отже, цей модуль представляє логіку виконання робочого завдання, яке отримує дані з різних джерел робочих місць і зберігає їх у відповідні колекції. Кожне джерело розпізнається за назвою, і відповідно до цього обирається відповідний парсер для обробки даних.

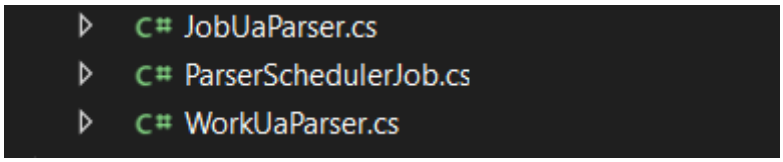


Рисунок 4.3. Відповідні класи які описують характеристики відповідних джерел для пошуку робочих місць.

4.3.2 Реалізація модулю аналізу зібраних даних

Наступний етап розробки був зумовлений тим що дані витягнуті з сайтів пошуку роботи потрібно було проаналізувати і розмістити в базі даних за логічними критеріями . Аналіз даних потрібен для виявлення корисних зв'язків, патернів та інформації в накопичених даних, що дозволяє отримати

цінні інсайти, приймати обґрунтовані рішення та покращувати швидкість виконання процесів. Оскільки попередній модуль має можливість автоматично планувати і виконувати завдання то був сенс зробити цю функцію і для цього модуля.

Отже цей модуль також має налаштоване середовище для виконання додатка, який використовує бібліотеку Quartz Scheduler. Після цього створюється клас AnalyzerJob , який реалізує інтерфейс IJob з бібліотеки Quartz. Цей клас відповідає за виконання аналізу даних.

У методі Execute класу відбувається виконання аналізу. За допомогою залежностей, таких як логер (ILogger) та сервіс аналізу (IAnalyzerService), отримуються завдання пошуку. Для кожного завдання, яке не було проаналізоване в поточний день, виконується сканування позицій і результати зберігаються.

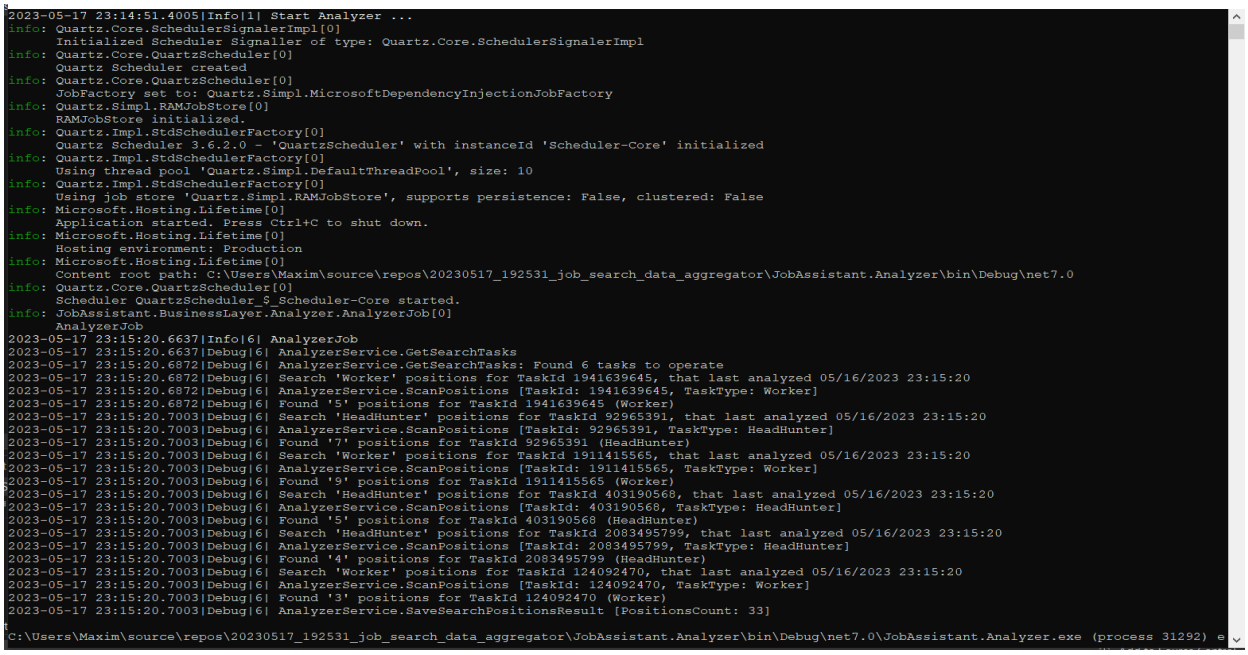


Рисунок 4.4. Відповідні повідомлення в журналі Analyzer

Після збереження результатів, відбувається затримка і завершується виконання аналізу.

Основна функція коду - виконання аналізу даних шляхом сканування позицій за заданими завданнями пошуку і збереження результатів аналізу які також записуються в журнал як на рисунку 4.4.

Ще одним важливим класом є AnalyzerService, який реалізує інтерфейс IAnalyzerService. Цей модуль виконує операції з аналізу та збереження результатів аналізу даних.

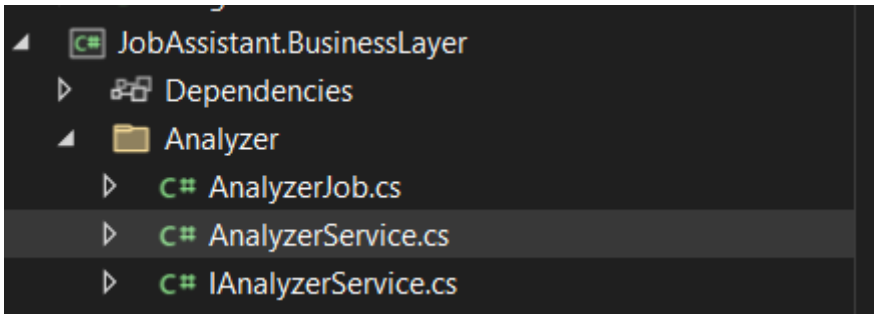


Рисунок 4.4. Класи що відповідають за функціонування модуля Analyzer

У конструкторі класу ініціалізуються залежності, такі як логер (ILogger) та сервіс доступу до даних аналізу (IAnalyzerDataService).

Модуль містить такі методи:

- GetSearchTasks: Отримує завдання пошуку, які потрібно аналізувати. Він викликає відповідний метод сервісу доступу до даних для отримання списку завдань.
- ScanPositions: Виконує сканування позицій на основі отриманого завдання пошуку. Залежно від типу завдання, виконується пошук робітників або вакансій за заданими критеріями. Результати аналізу упаковуються в об'єкти AnalyzerResult і повертаються як список.
- SaveSearchPositionsResult: Зберігає результати аналізу позицій, передані у вигляді списку AnalyzerResult. Викликає відповідний метод сервісу доступу до даних для збереження результатів.

Цей модуль коду відповідає за виконання основних операцій аналізу даних, отримання завдань пошуку, сканування позицій та збереження результатів аналізу.

4.3.3. Реалізація модулю відправки повідомлень

У даному модулі виконується налаштування логера (logger) та ініціалізація хоста (builder) для виконання сервісів.

Налаштування сервісів, пов'язаних з планувальником (Quartz), включають:

- Встановлення ідентифікатора планувальника (SchedulerId).
- Використання фабрики робіт на основі ін'єкції залежностей (UseMicrosoftDependencyInjectionJobFactory).
- Налаштування пам'яті для збереження даних планувальника (UseInMemoryStore).
- Використання типового пулу потоків (UseDefaultThreadPool) з обмеженням максимальної кількості одночасно виконуваних завдань (MaxConcurrency).
- Запланування виконання роботи (ScheduleJob) на основі класу NotificationSenderJob, з налаштуваннями часу початку (StartAt) та інтервалу виконання (WithDailyTimeIntervalSchedule).

Крім того, в модулі використовуються додаткові сервіси, додані через методи AddProviderServices та AddQuartzHostedService.

У підсумку, модуль встановлює логер, налаштовує планувальник для виконання роботи з використанням Quartz, та запускає хост для виконання сервісів. Основна робота модуля пов'язана з відправкою повідомлень.

Після цього клас клас **NotificationSenderJob**, який реалізує інтерфейс **IJob** і виконує завдання надсилання повідомлень.

Основний функціонал класу виконується у методі **Execute**, який приймає контекст виконання завдання **IJobExecutionContext**. У цьому методі відбувається наступне:

1. Створюється об'єкт **Stopwatch** для вимірювання часу виконання.
2. Логується початок надсилання підписних повідомлень.
3. Отримуються повідомлення для надсилання за допомогою служби **INotificationsDataService**.
4. Для кожного повідомлення викликається відповідний постачальник повідомлень залежно від типу повідомлення, отриманого в повідомленні. Постачальник повідомлень реалізує інтерфейс **IMessageProvider** і виконує надсилання повідомлення.
5. Після надсилання повідомлень виконується затримка за допомогою методу **Task.Delay** на випадковий проміжок часу.
6. Логується час, який зайняло надсилання повідомлень.
7. Повертається завершена задача за допомогою **Task.CompletedTask**.

Отже, цей клас відповідає за надсилання повідомлень з використанням постачальників повідомлень, які отримуються через фабрику **IMessageProviderFactory**.

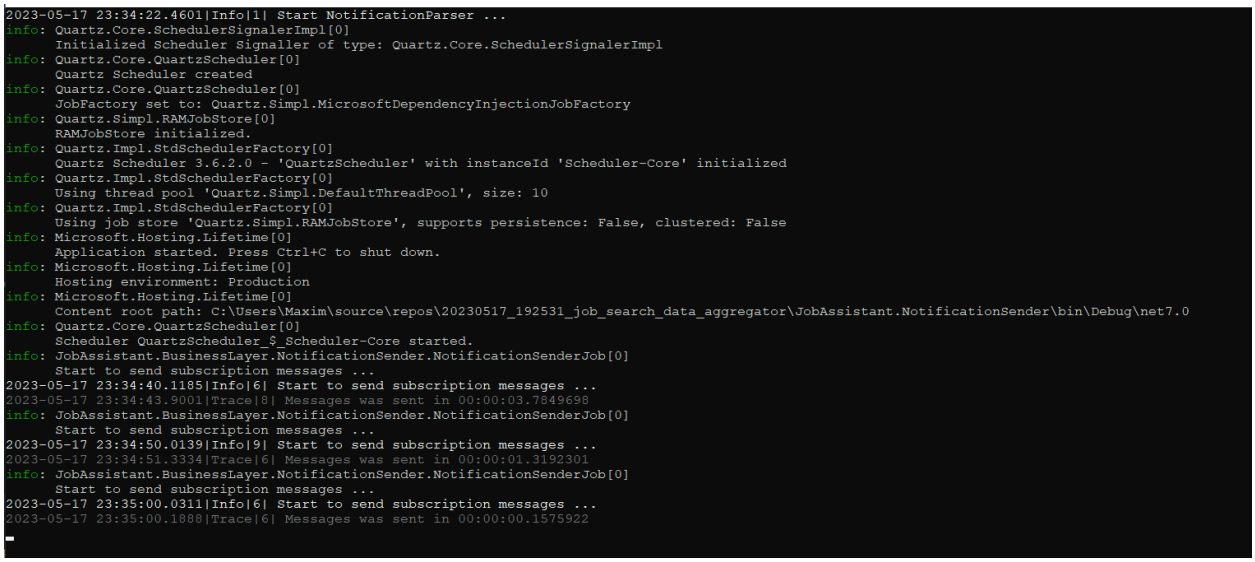


Рисунок 4.5. Відповідні повідомлення в журналі Notification

4.3.4. Реалізація Web частини

Реалізація WEB-частини полягала в поєднанні існуючого функціоналу в єдине ціле, і максимально зручне подання інформації для користувача.

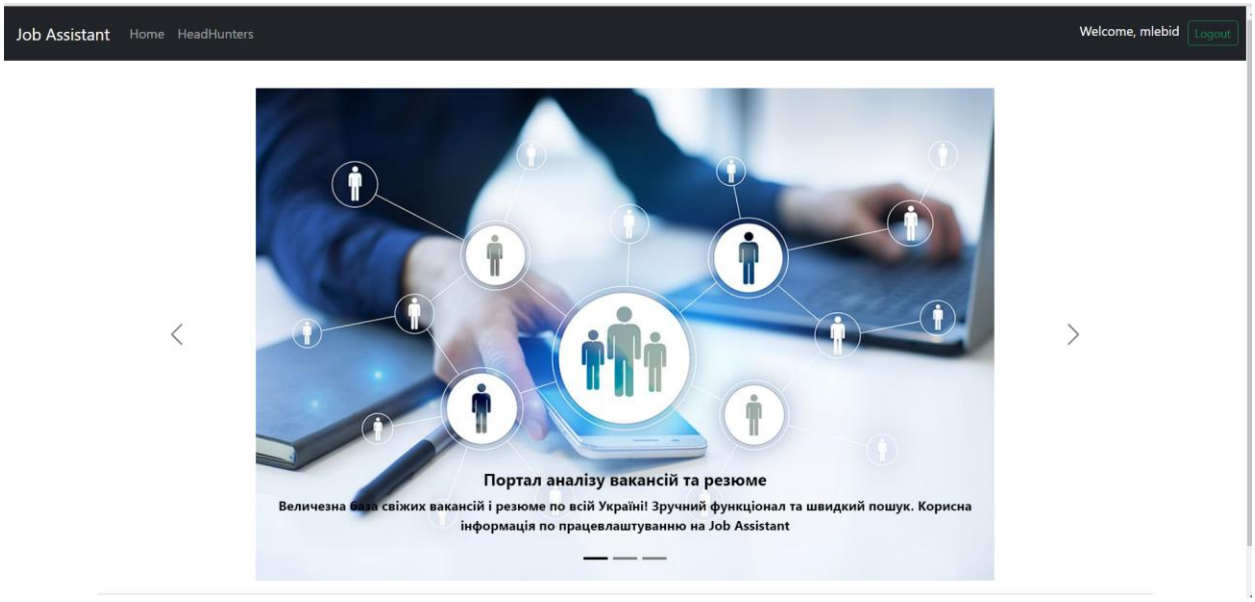


Рисунок 4.6. Головна сторінка додатку

На рисунку 4.6. видно що головна сторінка складається з короткої адаптивної інформації про сервіс, а також ряду кнопок які дозволяють перейти до поля авторизації/головної сторівнки та входу як на малюнку 4.6.

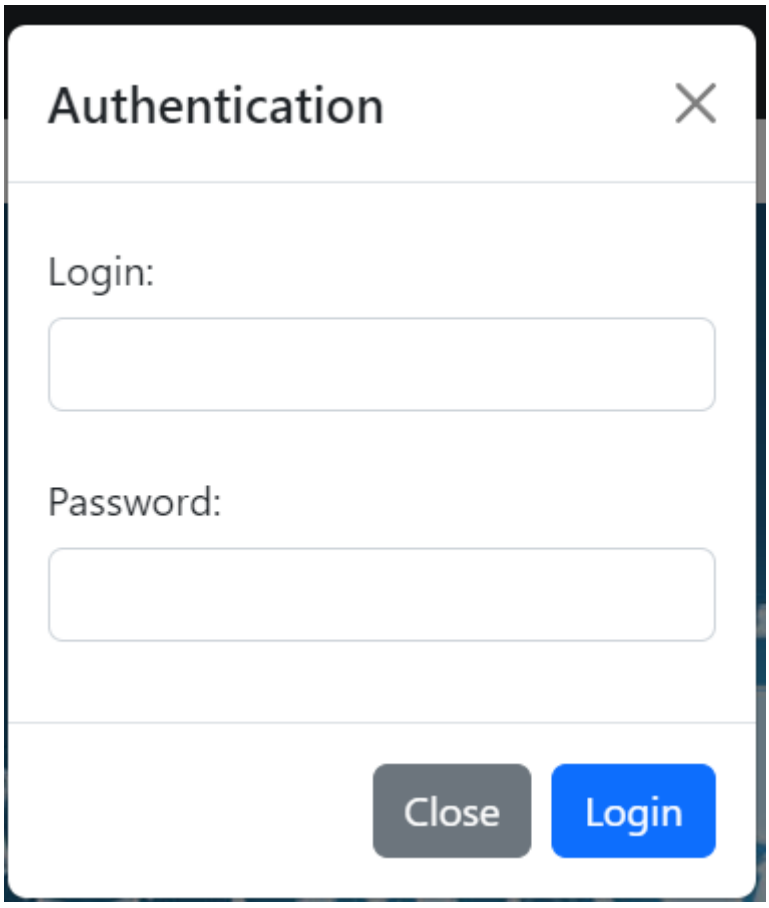


Рисунок 4.6. Форма авторизації

Після успішного входу користувачу відкривається додаткова вкладка в залежності від його ролі , сама вкладка складається з двох частин, мені зліва екрану, в якому будуть розміщатись запити які цікавлять користувачів, або про пошук цікавої вакансії або про пошук потрібного працівника, також була створена можливість щоб користувач заповнював одразу декілька запитів, на випадок якщо він хоче найняти або декількох працівників на різні посади, або його цікавлять декілька типів вакансій як на рисунку 4.7.

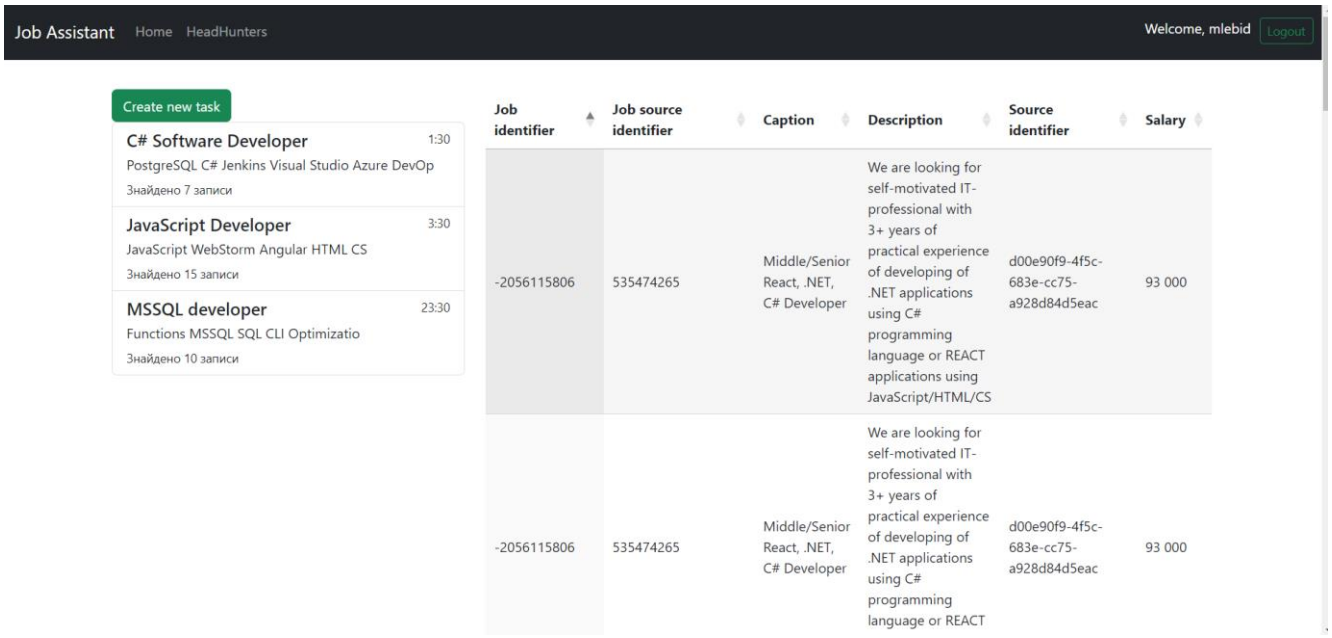


Рисунок 4.7. Вкладка для Workers

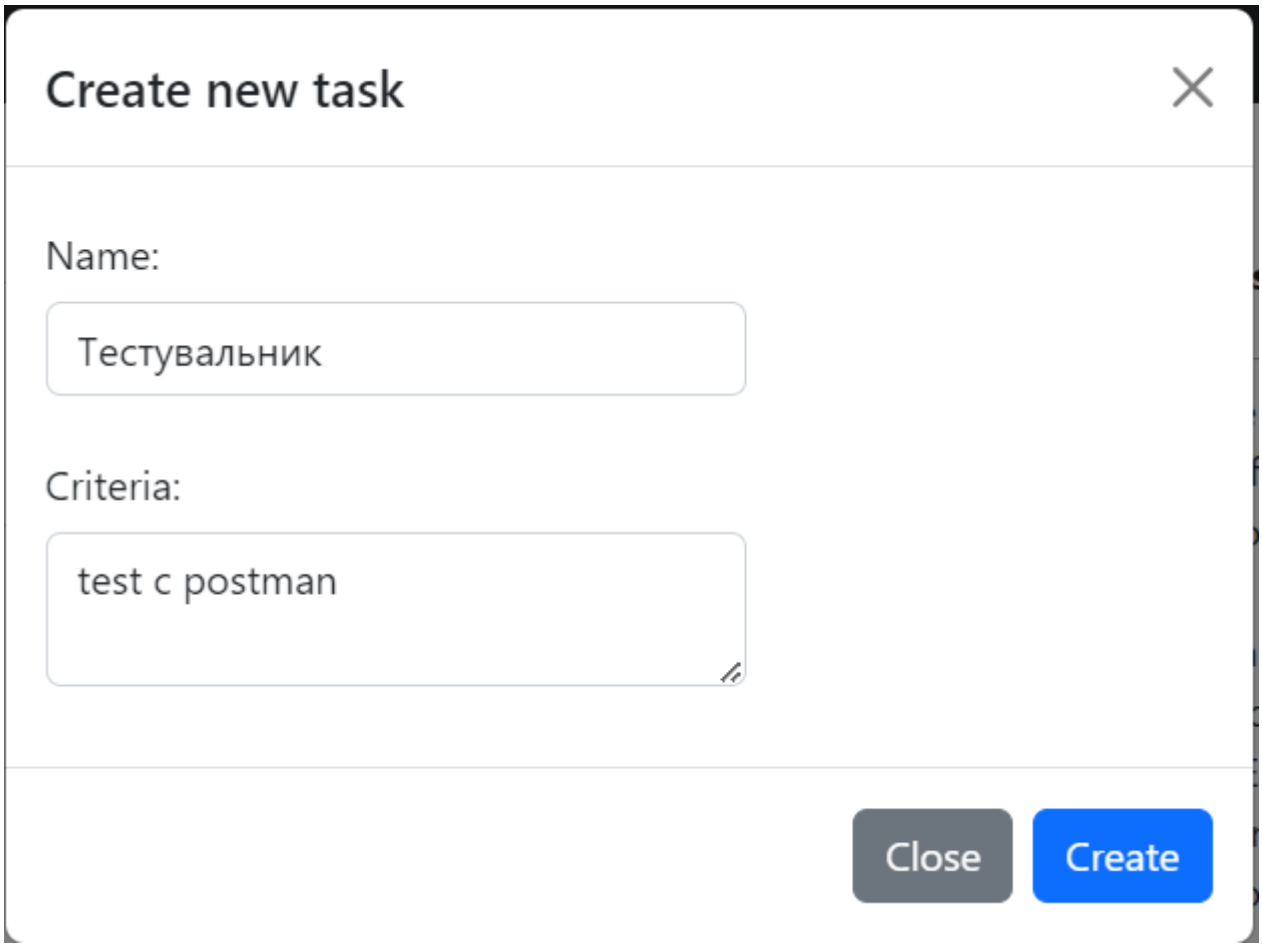


Рисунок 4.8. Форма створення нового запиту

4.4. Тестування

Тестування програмного забезпечення є процесом перевірки артефактів та поведінки програмного забезпечення з метою верифікації і перевірки. Воно надає об'єктивну та незалежну оцінку програмного забезпечення, що дозволяє бізнесу оцінити ризики його впровадження. Методи тестування включають аналіз вимог, перегляд архітектури та дизайну, співпрацю з розробниками для вдосконалення коду, виконання програми для перевірки поведінки, перевірку інфраструктури розгортання, участь у виробничій діяльності та інші методи.

Тестування програмного забезпечення надає об'єктивну та незалежну інформацію про якість програмного забезпечення та ризики його збою. Несправності та збої виникають через помилки, що робляться під час кодування програмного забезпечення. Не всі помилки призводять до збоїв, але навіть несправності, які не призводять до збоїв, можуть спричинити проблеми при зміні середовища. Прогалини вимог також можуть бути джерелом дефектів.

Тестування програмного забезпечення може бути статичним, коли перевіряються артефакти без виконання програми, або динамічним, коли виконується програма з тестовими випадками. Пасивне тестування означає перевірку поведінки системи без взаємодії з програмним продуктом. Дослідницьке тестування є підходом, який поєднує навчання, розробку тестів і їх виконання, щоб постійно оптимізувати якість роботи тестувальника.

Отже, тестування програмного забезпечення включає різні методи та підходи для перевевірки якості та поведінки програмного забезпечення. Це важливий процес, який допомагає забезпечити надійність та ефективність програмного продукту.

Під час розробки Web-частини застосунку кожна можлива функція була покрита тест кейсами відповідно до діаграми використання після чого за

результатами виконання сценаріїв дані були внесені в відповідну таблицю, після чого були виправлені знайдені недоліки , нижче наведено тест кейси для авторизації користувача

1. Позитивний сценарій - успішна авторизація:
- Вхідні дані: коректні облікові дані (правильне ім'я користувача і пароль).
- Очікуваний результат: користувач успішно авторизується і отримує доступ до системи.
2. Негативний сценарій - невірне ім'я користувача:
- Вхідні дані: невірне ім'я користувача, коректний пароль.
- Очікуваний результат: система відображає повідомлення про невірне ім'я користувача і не дозволяє авторизуватися.
3. Негативний сценарій - невірний пароль:
- Вхідні дані: коректне ім'я користувача, невірний пароль.
- Очікуваний результат: система відображає повідомлення про невірний пароль і не дозволяє авторизуватися.
4. Негативний сценарій - порожнє поле ім'я користувача:
- Вхідні дані: порожнє поле ім'я користувача, коректний пароль.
- Очікуваний результат: система відображає повідомлення про обов'язкове поле ім'я користувача і не дозволяє авторизуватися.
5. Негативний сценарій - порожнє поле пароля:
- Вхідні дані: коректне ім'я користувача, порожнє поле пароля.
- Очікуваний результат: система відображає повідомлення про обов'язкове поле

Для додаткової перевірки програми було здійснено тестування за допомогою методів чорного ящика та білого ящика. Це тестування дозволило перевірити функціональність додатку і виправити виявлені помилки у програмному коді.

Під час тестування використовувалися методи граничних значень, класів еквівалентності, а також стохастичне тестування. Також було проведене структурне тестування коду програми за допомогою методів маршрутного тестування.

ВИСНОВКИ

У дипломній роботі виконано розробку Web-платформи для агрегації даних сайтів пошуку роботи

У процесі створення платформи було проаналізовано принципи створення автоматичного планування і виконування завдань а також основні особливості створення веб-додатків за допомогою Visual Studio мовою програмування C#. Було виявлено певні проблеми парсингу даних.

Під час написання дипломної роботи було проведено аналіз засобів web-платформ. Було покращено навички роботи з бібліотеками для логування, та бібліотеками автоматизації задач .

Під час виконання дипломної роботи були виконані всі поставлені завдання:

1. Проведено аналіз аналогів в схожих Web-платформах;
2. Проаналізовано технічні засоби та вибір відповідних для розробки;
3. Розроблено вимоги до web-платформи на основі аналізу переваг і недоліків аналогів;
4. Спроектовано і розроблено web-платформу;
5. Проведено тестування платформи;

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ринок праці 2023 [Електронний ресурс] – Режим доступу до ресурсу: https://www.epravda.com.ua/publications/2023/01/25/696322/

2. What is Data Aggregation? [Електронний ресурс] – Режим доступу до ресурсу: hhttps://www.tibco.com/reference-center/what-is-data-aggregation

3. Get started with ASP.NET Core MVC [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-7.0&tabs=visual-studio

4. What is PostgreSQL? [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/about/

5. What's new in Visual Studio 2022? [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/visualstudio/ide/whats-new-visual-studio-2022?view=vs-2022

6. NLog documentation [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/nlog/nlog/wiki

7. Quartz.NET [Електронний ресурс] – Режим доступу до ресурсу: https://www.quartz-scheduler.net/

8. Entity Relationship Diagram (ERD) [Електронний ресурс] – Режим доступу до ресурсу: https://www.smartdraw.com/entity-relationship-diagram/

9. What is Package Diagram? [Електронний ресурс] – Режим доступу до ресурсу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/

10. UML Use Case Diagram [Електронний ресурс] – Режим доступу до ресурсу: https://www.lucidchart.com/pages/uml-use-case-diagram

11. What is the Agile methodology? [Електронний ресурс] – Режим доступу до ресурсу: https://www.atlassian.com/agile

12. Kanban vs. scrum: which agile are you? [Електронний ресурс] – Режим

доступу до ресурсу: https://www.atlassian.com/agile/kanban/kanban-vs-scrum

ДОДАТОК А
Демонстраційні матеріали

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка Web-платформи для агрегації даних сайтів пошуку роботи мовою C#

Виконав студент 4 курсу
Групи ПД-41
Лебідь Максим Максимович
Керівник роботи
к.т.н., доц., доцент кафедри ІПЗ Трінтіна Н.А.

Київ – 2023

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** спрощення процесу агрегації даних з популярних сайтів пошуку роботи за рахунок його автоматизації з використанням веб-додатку мовою C#
- **Об’єкт дослідження** процес агрегації даних з популярних сайтів пошуку роботи .
- **Предмет дослідження** технології розробки Web-додаток для агрегації даних з популярних сайтів пошуку роботи .

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Аналіз аналогів в схожих Web-платформах.
2. Аналіз технічних засобів та вибір відповідних.
3. Розробка вимог до web-платформи на основі аналізу переваг і недоліків аналогів.
4. Проектування і розробка web-платформи.
5. Тестування.

3

АНАЛІЗ АНАЛОГІВ

Показник	Indeed	Glassdoor	Єдиний портал вакансій	JobAssistant
Велика кількість вакансій:	+	-	+	+
Покриття географії	+	-	+	+
Фільтрація та розширені пошукові функції	+	+	-	+
Інформація про компанії:	-	+	-	-
Перегляд статистики по вакансіям за проміжки часу	-	-	-	+
Сповіщання користувачів по цікавим для них позиціях при додаванні вакансій для них	-	-	-	+
Платні пакети та підписки:	+	+	-	-
Спеціалізація на певних індустріях	-	+	-	-
Співпраця з роботодавцями	-	-	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1. Пошук вакансій на різних сайтах пошуку роботи.
2. Веб-інтерфейс для користувачів, який дозволяє шукати вакансії та налаштовувати параметри пошуку.
3. Фільтрація та сортування результатів пошуку за різними критеріями.
4. Автоматичне оновлення даних про вакансії зі сторонніх сайтів пошуку роботи.
5. Створення облікових записів користувачів та збереження їх налаштувань пошуку.
6. Розробка веб-платформи з використанням мови C# та фреймворку ASP.NET.

5

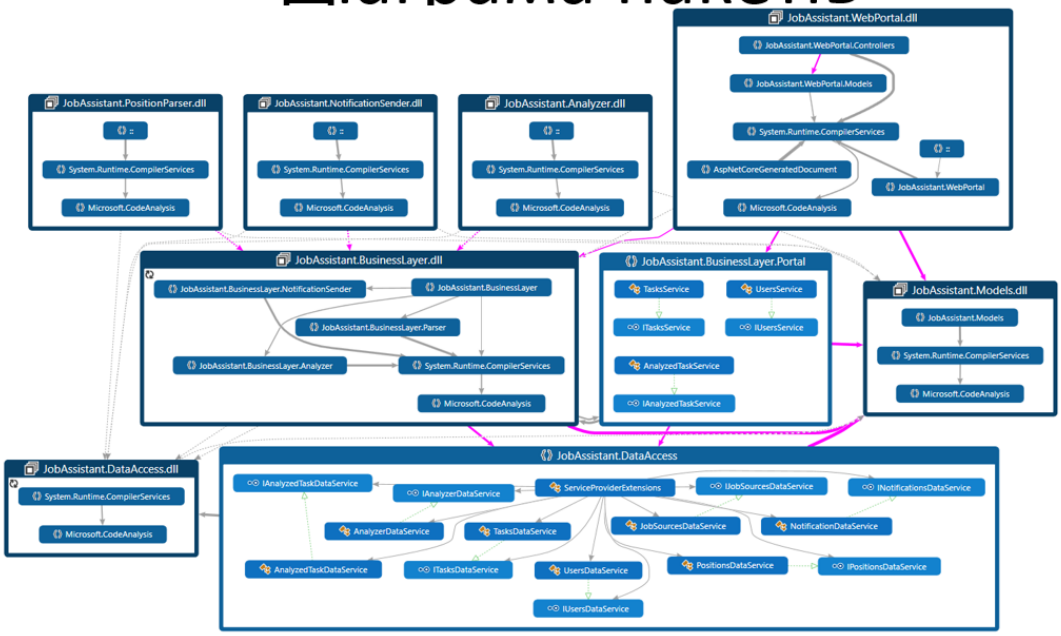
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

Діаграма варіантів використання

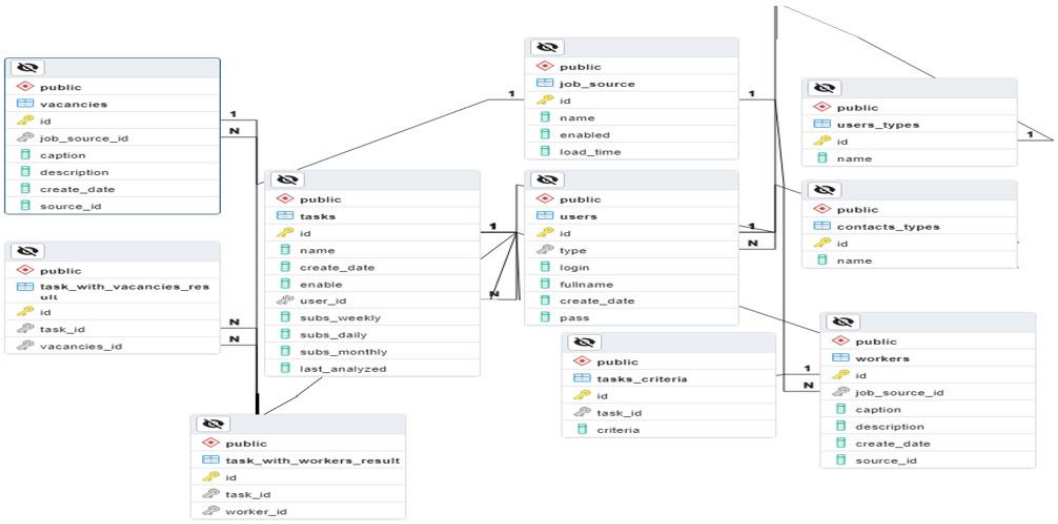
7

Діаграма пакетів



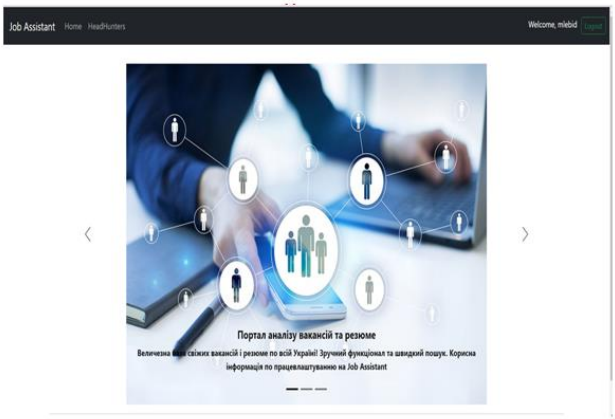
8

СХЕМА БД

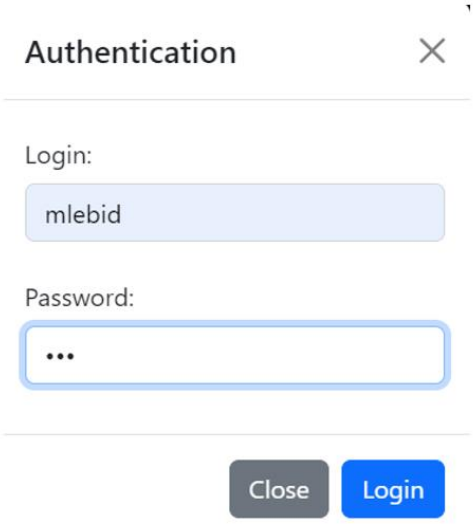


9

ЕКРАННІ ФОРМИ



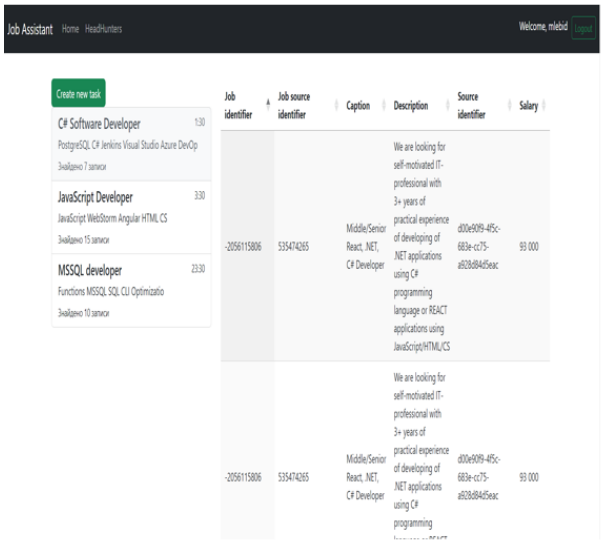
Головна сторінка



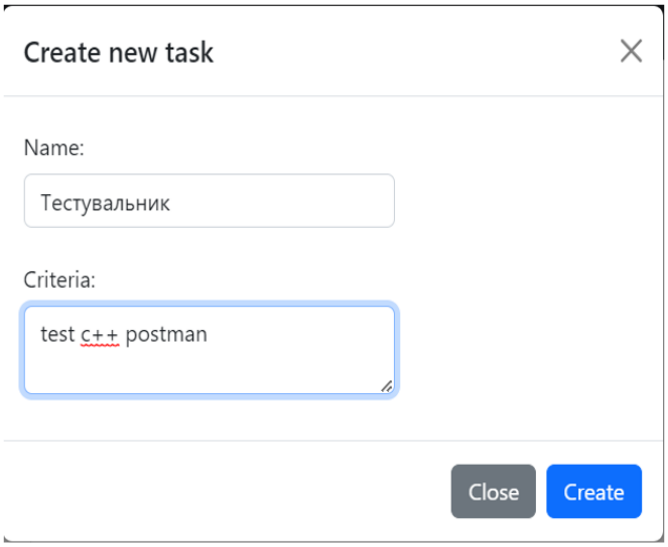
Форма авторизації

10

ЕКРАННІ ФОРМИ



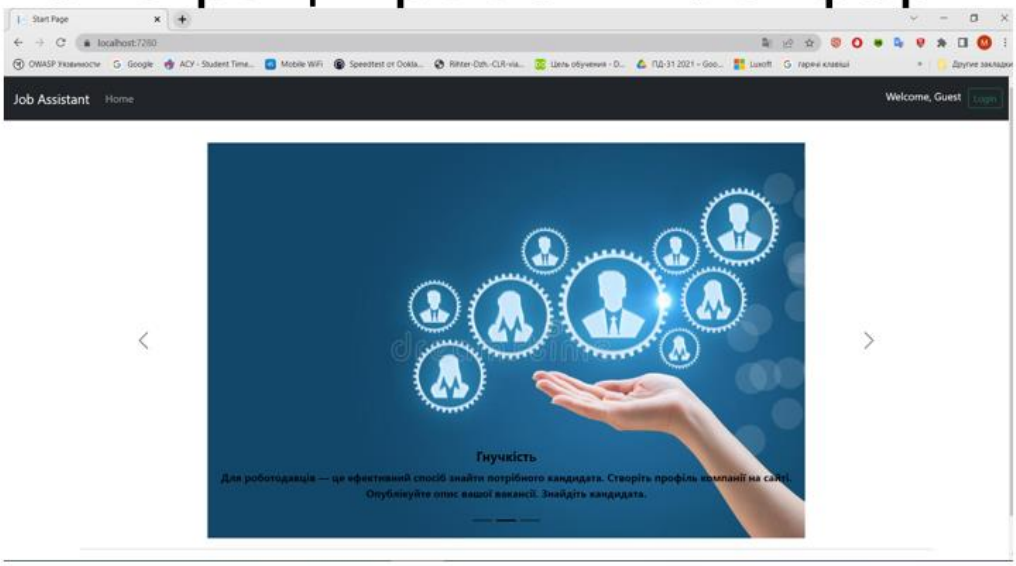
Вкладка користувача



Форма створення запитів

11

Демонстрація роботи платформи



12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Лебідь М.М. ASP.NET FRAMEWORK MVC/ Трінтіна Н.А, Лебідь М.М. // СУЧАСНИЙ СТАН ТА ПЕРСПЕКТИВИ РОЗВИТКУ ІОТ, 7 квітня 2023р., ДУТ, м. Київ - К: ДУТ, 2023. –с. 62

2. Лебідь М.М. РОЗРОБКА WEB-ПЛАТФОРМИ ДЛЯ АГРЕГАЦІЇ ДАНИХ/ Трінтіна Н.А, Лебідь М.М. // Застосування програмного забезпечення в інфокомунікаційних технологіях, 20 квітня 2023р., ДУТ, м. Київ - К: ДУТ, 2023. –с. 87

12

ВИСНОВКИ

1.Проведено аналіз аналогів в схожих Web-платформах.
2.Проаналізовано технічні засоби та вибір відповідних для розробки.
3.Розроблено вимоги до web-платформи на основі аналізу переваг і недоліків аналогів.
4.Спроектовано і розроблено web-платформу.
5.Проведено тестування платформи.

13

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б
Основні блоки коду

```
namespace JobAssistant.Models;

/// <summary>
/// The analyzer command class
/// </summary>
public class AnalyzerCommand
{
    /// <summary>
    /// Gets or sets the value of the task id
    /// </summary>
    public int TaskId { get; set; }

    /// <summary>
    /// Gets or sets the value of the task type
    /// </summary>
    public string TaskType { get; set; }

    /// <summary>
    /// Gets or sets the value of the criteries
    /// </summary>
    public List<string> Criteries { get; set; }

    /// <summary>
    /// Gets or sets the value of the last type analyzed
    /// </summary>
    public DateTime LastTypeAnalyzed { get; set; }
}

namespace JobAssistant.Models;

/// <summary>
/// The analyzer result class
/// </summary>
public class AnalyzerResult
{
    /// <summary>
    /// Gets or sets the value of the task id
    /// </summary>
    public int TaskId { get; set; }

    /// <summary>
    /// Gets or sets the value of the position id
    /// </summary>
    public int PositionId { get; set; }
}

namespace JobAssistant.Models;

/// <summary>
/// The contact type class
/// </summary>
public class ContactType
{
    /// <summary>
    /// Gets or sets the value of the id
    /// </summary>
    public int Id { get; set; }

    /// <summary>
```

```
    /// Gets or sets the value of the name
    /// </summary>
    public string Name { get; set; }
}

namespace JobAssistant.Models;

/// <summary>
/// The job source class
/// </summary>
public class JobSource
{
    /// <summary>
    /// Gets or sets the value of the id
    /// </summary>
    public int Id { get; set; }

    /// <summary>
    /// Gets or sets the value of the name
    /// </summary>
    public string Name { get; set; }

    /// <summary>
    /// Gets or sets the value of the enabled
    /// </summary>
    public bool Enabled { get; set; }

    /// <summary>
    /// Gets or sets the value of the load time
    /// </summary>
    public DateTime LoadTime { get; set; }
}

namespace JobAssistant.Models;

/// <summary>
/// The position class
/// </summary>

public class Position
{
    /// <summary>
    /// Initializes a new instance of the <see cref="Position"/> class
    /// </summary>
    /// <param name="id">The id</param>
    /// <param name="jobSourceId">The job source id</param>
    /// <param name="caption">The caption</param>
    /// <param name="description">The description</param>
    /// <param name="sourceId">The source id</param>
    /// <param name="salary">The salary</param>
    public Position(int id, int jobSourceId, string caption, string description, string
sourceId, string salary)
    {
        Id = id;
        JobSourceId = jobSourceId;
        Caption = caption;
        Description = description;
        SourceId = sourceId;
        Salary = salary;
    }

    /// <summary>
    /// Gets or sets the value of the id
```

```
    /// </summary>
    public int Id { get; set; }

    /// <summary>
    /// Gets or sets the value of the job source id
    /// </summary>
    public int JobSourceId { get; set; }

    /// <summary>
    /// Gets or sets the value of the caption
    /// </summary>
    public string Caption { get; set; }

    /// <summary>
    /// Gets or sets the value of the description
    /// </summary>
    public string Description { get; set; }

    /// <summary>
    /// Gets or sets the value of the create date
    /// </summary>
    public DateTime CreateDate { get; set; }

    /// <summary>
    /// Gets or sets the value of the source id
    /// </summary>
    public string SourceId { get; set; }

    /// <summary>
    /// Gets or sets the value of the salary
    /// </summary>
    public string Salary { get; set; }
}

namespace JobAssistant.Models;

/// <summary>
/// The subscription message class
/// </summary>
public class SubscriptionMessage
{
    /// <summary>
    /// Gets or sets the value of the message type
    /// </summary>
    public string MessageType { get; set; }

    /// <summary>
    /// Gets or sets the value of the message
    /// </summary>
    public string Message { get; set; }

    /// <summary>
    /// Gets or sets the value of the receiver
    /// </summary>
    public string Receiver { get; set; }
}

namespace JobAssistant.Models;

/// <summary>
/// The task details class
/// </summary>
public class TaskDetails
```

```
{
    /// <summary>
    /// Initializes a new instance of the <see cref="TaskDetails"/> class
    /// </summary>
    /// <param name="taskId">The task id</param>
    /// <param name="name">The name</param>
    /// <param name="criteries">The criteries</param>
    /// <param name="positionsFound">The positions found</param>
    /// <param name="lastAnalyzed">The last analyzed</param>
    public TaskDetails(int taskId, string name, List<string> criteries, int
positionsFound, DateTime? lastAnalyzed)
    {
        Name = name;
        Criteries = criteries;
        TaskId = taskId;
        PositionsFound = positionsFound;
        LastAnalyzed = lastAnalyzed;
    }

    /// <summary>
    /// Gets or sets the value of the task id
    /// </summary>
    public int TaskId { get; set; }

    /// <summary>
    /// Gets or sets the value of the name
    /// </summary>
    public string Name { get; set; }

    /// <summary>
    /// Gets or sets the value of the last analyzed
    /// </summary>
    public DateTime? LastAnalyzed { get; set; }

    /// <summary>
    /// Gets or sets the value of the criteries
    /// </summary>
    public List<string> Criteries { get; set; }

    /// <summary>
    /// Gets or sets the value of the positions found
    /// </summary>
    public int PositionsFound { get; set; }
}

namespace JobAssistant.Models;

/// <summary>
/// The user class
/// </summary>
public class User
{
    /// <summary>
    /// Initializes a new instance of the <see cref="User"/> class
    /// </summary>
    /// <param name="login">The login</param>
    /// <param name="fullName">The full name</param>
    /// <param name="type">The type</param>
    /// <param name="password">The password</param>
    public User(string login, string fullName, string type, string password)
    {
        Login = login;
        FullName = fullName;
        Type = type;
```

```
        Password = password;
    }

    /// <summary>
    /// Gets or inits the value of the login
    /// </summary>
    public string Login { get; init; }

    /// <summary>
    /// Gets or inits the value of the full name
    /// </summary>
    public string FullName { get; init; }

    /// <summary>
    /// Gets or inits the value of the type
    /// </summary>
    public string Type { get; init; }

    /// <summary>
    /// Gets or sets the value of the password
    /// </summary>
    public string Password { get; set; }
}

namespace JobAssistant.Models;

/// <summary>
/// The vacancy class
/// </summary>
/// <seealso cref="Position"/>
public class Vacancy : Position
{
    /// <summary>
    /// Initializes a new instance of the <see cref="Vacancy"/> class
    /// </summary>
    /// <param name="id">The id</param>
    /// <param name="jobSourceId">The job source id</param>
    /// <param name="caption">The caption</param>
    /// <param name="description">The description</param>
    /// <param name="sourceId">The source id</param>
    /// <param name="salary">The salary</param>
    public Vacancy(int id, int jobSourceId, string caption, string description, string
sourceId, string salary) :
        base(id, jobSourceId, caption, description, sourceId, salary)
    {
    }
}

namespace JobAssistant.Models;

/// <summary>
/// The worker class
/// </summary>
/// <seealso cref="Position"/>
public class Worker : Position
{
    /// <summary>
    /// Initializes a new instance of the <see cref="Worker"/> class
    /// </summary>
    /// <param name="id">The id</param>
    /// <param name="jobSourceId">The job source id</param>
    /// <param name="caption">The caption</param>
    /// <param name="description">The description</param>
```

```
    /// <param name="sourceId">The source id</param>
    /// <param name="salary">The salary</param>
    public Worker(int id, int jobSourceId, string caption, string description, string
sourceId, string salary) :
        base(id, jobSourceId, caption, description, sourceId, salary)
    {
    }
}

using JobAssistant.BusinessLayer.Analyzer;
using JobAssistant.BusinessLayer.NotificationSender;
using JobAssistant.BusinessLayer.Parser;
using JobAssistant.BusinessLayer.Portal;
using JobAssistant.DataAccess;
using Microsoft.Extensions.DependencyInjection;

namespace JobAssistant.BusinessLayer;

public static class ServiceProviderExtensions
{
    public static void AddProviderServices(this IServiceCollection services)
    {
        services.AddProviderDataServices();

        services.AddSingleton<IExternalSourceParser, FakePositionGenerator>();

        services.AddSingleton<IMessageProviderFactory, ProvidersFactory>();

        services.AddSingleton<IAnalyzerService, AnalyzerService>();

        services.AddTransient<IUsersService, UsersService>();

        services.AddTransient<ITasksService, TasksService>();

        services.AddTransient<IAnalyzedTaskService, AnalyzedTaskService>();
    }
}


using JobAssistant.Models;

namespace JobAssistant.BusinessLayer.Portal;

public interface ITasksService
{
    Task<List<TaskDetails>> GetTasksByUser(string userLogin);
}

using JobAssistant.DataAccess;
using JobAssistant.Models;
using Microsoft.Extensions.Logging;

namespace JobAssistant.BusinessLayer.Portal;

public class TasksService : ITasksService
{
    private readonly ILogger<TasksService> _logger;

    private readonly ITasksDataService _dataService;

    public TasksService(ILogger<TasksService> logger, ITasksDataService dataService)
    {
        _logger = logger;
```

```
        _dataService = dataService;
    }

    public async Task<List<TaskDetails>> GetTasksByUser(string userLogin)
    {
        if (string.IsNullOrWhiteSpace(userLogin))
        {
            return await Task.FromResult(new List<TaskDetails>());
        }

        var result = await _dataService.GetTasksByUser(userLogin);

        _logger.LogDebug("TasksService.GetTasksByUser Returned {Count} entries by
{Login}", result.Count, userLogin);

        return result;
    }
}
```

```
using JobAssistant.Models;

namespace JobAssistant.BusinessLayer.Portal;

public interface IUsersService
{
    Task<User?> Login(string login, string password);
}
```

```
using JobAssistant.DataAccess;
using JobAssistant.Models;
using Microsoft.Extensions.Logging;

namespace JobAssistant.BusinessLayer.Portal;

public class UsersService : IUsersService
{
    private readonly ILogger<UsersService> _logger;

    private readonly IUsersDataService _dataService;

    public UsersService(ILogger<UsersService> logger, IUsersDataService dataService)
    {
        _logger = logger;
        _dataService = dataService;
    }

    public async Task<User?> Login(string login, string password)
    {
        if (string.IsNullOrWhiteSpace(login) ||
            string.IsNullOrWhiteSpace(password))
        {
            return null;
        }

        if (login.Length > 10 || password.Length > 30)
        {
            return null;
        }

        var user = await _dataService.Login(login, password);

        if (user != null)
        {
```

```
            _logger.LogDebug("User {Login}, {FullName} authenticated", user.Login,
user.FullName);
        }

        return user;
    }
}
```

```
using JobAssistant.Models;

namespace JobAssistant.BusinessLayer.Portal;

public interface IAnalyzedTaskService
{
    Task<List<Position>> GetTaskPositions(int taskId);
}
```

```
using JobAssistant.DataAccess;
using JobAssistant.Models;
using Microsoft.Extensions.Logging;

namespace JobAssistant.BusinessLayer.Portal;

public class AnalyzedTaskService : IAnalyzedTaskService
{
    private readonly ILogger<AnalyzedTaskService> _logger;

    private readonly IAnalyzedTaskDataService _dataService;

    public AnalyzedTaskService(IAnalyzedTaskDataService dataService,
ILogger<AnalyzedTaskService> logger)
    {
        _dataService = dataService;
        _logger = logger;
    }

    public async Task<List<Position>> GetTaskPositions(int taskId)
    {
        _logger.LogDebug("AnalyzedTaskService.GetTaskPositions [TaskId: {TaskId}]",
taskId);

        return await _dataService.GetTaskPositions(taskId);
    }
}
```

```
using System.Diagnostics;
using JobAssistant.DataAccess;
using JobAssistant.Models;
using Microsoft.Extensions.Logging;
using Quartz;

namespace JobAssistant.BusinessLayer.Parser;

public class ParserSchedulerJob : IJob
{
    private readonly IJobSourcesDataService _jobSourcesDataService;

    private readonly IPositionsDataService _positionsDataService;

    private readonly ILogger<ParserSchedulerJob> _logger;
```

```
    public ParserSchedulerJob(IJobSourcesDataService jobSourcesDataService,
        IPositionsDataService positionsDataService, ILogger<ParserSchedulerJob> logger)
    {
        _jobSourcesDataService = jobSourcesDataService;

        _positionsDataService = positionsDataService;

        _logger = logger;
    }

    public async Task Execute(IJobExecutionContext context)
    {
        var sw = new Stopwatch();

        sw.Start();

        _logger.LogInformation("Start to parse position sources ...");

        var positionSources = _jobSourcesDataService.GetJobSources();

        var parsedVacancies = new List<Vacancy>();

        var parsedWorkers = new List<Worker>();

        foreach (var source in positionSources.Where(t => t.Enabled))
        {
            switch (source.Name)
            {
                case "Job.ua":
                    var jobUaParser = new JobUaParser();

                    parsedVacancies.AddRange(jobUaParser.LoadVacancies(source.LoadTime));
                    parsedWorkers.AddRange(jobUaParser.LoadWorkers(source.LoadTime));
                    break;
                case "hh.ua":
                    var hParser = new HhUaParser();

                    parsedVacancies.AddRange(hParser.LoadVacancies(source.LoadTime));

                    parsedWorkers.AddRange(hParser.LoadWorkers(source.LoadTime));
                    break;
                case "Work.ua":
                    var workParser = new HhUaParser();

                    parsedVacancies.AddRange(workParser.LoadVacancies(source.LoadTime));

                    parsedWorkers.AddRange(workParser.LoadWorkers(source.LoadTime));
                    break;
                case "djinni.ua":
                    var djinniParser = new HhUaParser();


parsedVacancies.AddRange(djinniParser.LoadVacancies(source.LoadTime));

                    parsedWorkers.AddRange(djinniParser.LoadWorkers(source.LoadTime));
                    break;
            }
        }

        await Task.Delay(new Random(DateTime.Now.Millisecond).Next(100, 4000));

        _logger.LogDebug("Parsed {Count} vacancies", parsedVacancies.Count);
```

```
        _logger.LogDebug("Parsed {Count} workers", parsedWorkers.Count);

        _positionsDataService.SaveVacancies(parsedVacancies);

        _positionsDataService.SaveWorkers(parsedWorkers);

        _logger.LogTrace("Parse job completed in {Elapsed}", sw.Elapsed);
    }
}
```

```
using JobAssistant.Models;

namespace JobAssistant.BusinessLayer.Parser;

public interface IExternalSourceParser
{
    List<Vacancy> LoadVacancies(DateTime fromPeriod);

    List<Worker> LoadWorkers(DateTime fromPeriod);
}
```

```
using Bogus;
using JobAssistant.Models;

namespace JobAssistant.BusinessLayer.Parser;

public class FakePositionGenerator : IExternalSourceParser
{
    private readonly Faker _faker = new ("uk")
    {
        Random = new Randomizer(DateTime.Now.Millisecond)
    };

    public List<Vacancy> LoadVacancies(DateTime fromPeriod)
    {
        var foundVacanciesCount = new Random().Next(10, 50);

        var result = new List<Vacancy>(foundVacanciesCount);

        for (var i = 0; i < foundVacanciesCount; i++)
        {
            result.Add(new Vacancy(_faker.Random.Int(), _faker.Random.Int(),
_faker.Name.JobTitle(),
                _faker.Name.JobDescriptor(), _faker.Random.Guid().ToString(),
RandomSalary()));
        }

        return result;
    }

    public List<Worker> LoadWorkers(DateTime fromPeriod)
    {
        var foundVacanciesCount = new Random().Next(10, 50);

        var result = new List<Worker>(foundVacanciesCount);

        for (var i = 0; i < foundVacanciesCount; i++)
        {
            result.Add(new Worker(_faker.Random.Int(), _faker.Random.Int(),
_faker.Name.JobTitle(),
```

```
                _faker.Name.JobDescriptor(), _faker.Random.Guid().ToString(),
RandomSalary()));
        }

        return result;
    }

    private string RandomSalary()
    {
        var thousandPart = new Random().Next(30, 150);

        return $"{thousandPart} 000";
    }
}


namespace JobAssistant.BusinessLayer.NotificationSender;

public class ProvidersFactory : IMessageProviderFactory
{
    private readonly EmailSender _emailSender;

    private readonly TelegramSender _telegramSender;

    public ProvidersFactory()
    {
        _emailSender = new EmailSender();

        _telegramSender = new TelegramSender();
    }

    public IMessageProvider GetMessageProvider(string contactType)
    {
        switch (contactType)
        {
            case "Email":
                return _emailSender;
            case "Telegram":
                return _telegramSender;
        }

        throw new ArgumentException($"Unknown contact type '{contactType}'");
    }
}


using System.Diagnostics;
using JobAssistant.DataAccess;
using Microsoft.Extensions.Logging;
using Quartz;

namespace JobAssistant.BusinessLayer.NotificationSender;

public class NotificationSenderJob : IJob
{
    private readonly ILogger<NotificationSenderJob> _logger;

    private readonly INotificationsDataService _dataService;

    private readonly IMessageProviderFactory _providerFactory;

    public NotificationSenderJob(ILogger<NotificationSenderJob> logger,
INotificationsDataService dataService,
```

```
        IMessageProviderFactory providerFactory)
    {
        _logger = logger;

        _dataService = dataService;

        _providerFactory = providerFactory;
    }

    public async Task Execute(IJobExecutionContext context)
    {
        var sw = new Stopwatch();

        sw.Start();

        _logger.LogInformation("Start to send subscription messages ...");

        var messagesToSend = _dataService.GetMessagesForSend();

        foreach (var message in messagesToSend)
        {
            var sender = _providerFactory.GetMessageProvider(message.MessageType);

            sender.Send(message);
        }

        await Task.Delay(new Random(DateTime.Now.Millisecond).Next(100, 4000));

        _logger.LogTrace("Messages was sent in {Elapsed}", sw.Elapsed);

        await Task.CompletedTask;
    }
}
```

```
using JobAssistant.Models;

namespace JobAssistant.BusinessLayer.NotificationSender;

public interface IMessageProvider
{
    void Send(SubscriptionMessage message);
}
```

```
namespace JobAssistant.BusinessLayer.NotificationSender;

public interface IMessageProviderFactory
{
    IMessageProvider GetMessageProvider(string contactType);
}
```

```
using JobAssistant.Models;

namespace JobAssistant.BusinessLayer.NotificationSender;

public interface IMessageSender
{
    void SendSubscriptionMessages(List<SubscriptionMessage> messages);
}
```

```
using JobAssistant.Models;
```

```
namespace JobAssistant.BusinessLayer.Analyzer;

public interface IAnalyzerService
{
    List<AnalyzerCommand> GetSearchTasks();

    List<AnalyzerResult> ScanPositions(AnalyzerCommand command);

    void SaveSearchPositionsResult(List<AnalyzerResult> positions);
}
```

```
using System.Diagnostics;
using JobAssistant.Models;
using Microsoft.Extensions.Logging;
using Quartz;

namespace JobAssistant.BusinessLayer.Analyzer;

public class AnalyzerJob : IJob
{
    private readonly ILogger<AnalyzerJob> _logger;

    private readonly IAnalyzerService _analyzerService;

    public AnalyzerJob(ILogger<AnalyzerJob> logger, IAnalyzerService analyzerService)
    {
        _logger = logger;

        _analyzerService = analyzerService;
    }

    public async Task Execute(IJobExecutionContext context)
    {
        _logger.LogInformation("AnalyzerJob");

        var sw = new Stopwatch();

        sw.Start();

        var tasks = _analyzerService.GetSearchTasks();

        var newPositions = new List<AnalyzerResult>();

        foreach (var task in tasks.Where(t=>t.LastTypeAnalyzed < DateTime.Now.Date))
        {
            _logger.LogDebug(
                "Search '{TaskType}' positions for TaskId {TaskId}, that last analyzed
{LastAnalyzed}",
                task.TaskType, task.TaskId, task.LastTypeAnalyzed);

            var taskPositions = _analyzerService.ScanPositions(task);

            newPositions.AddRange(taskPositions);

            _logger.LogDebug("Found '{Count}' positions for TaskId {TaskId}
({TaskType})",
                taskPositions.Count, task.TaskId, task.TaskType);
        }

        _analyzerService.SaveSearchPositionsResult(newPositions);

        await Task.Delay(new Random(DateTime.Now.Millisecond).Next(100, 4000));
```

```
        _logger.LogTrace("Analyze has completed in {Elapsed}", sw.Elapsed);
    }
}

using JobAssistant.DataAccess;
using JobAssistant.Models;
using Microsoft.Extensions.Logging;

namespace JobAssistant.BusinessLayer.Analyzer;

public class AnalyzerService : IAnalyzerService
{
    private readonly ILogger<AnalyzerService> _logger;

    private readonly IAnalyzerDataService _dataService;

    public AnalyzerService(ILogger<AnalyzerService> logger, IAnalyzerDataService
dataService)
    {
        _logger = logger;

        _dataService = dataService;
    }

    public List<AnalyzerCommand> GetSearchTasks()
    {
        _logger.LogDebug("AnalyzerService.GetSearchTasks");

        var result = _dataService.GetSearchTasks();

        _logger.LogDebug("AnalyzerService.GetSearchTasks: Found {Count} tasks to
operate", result.Count);

        return result;
    }

    public List<AnalyzerResult> ScanPositions(AnalyzerCommand command)
    {
        _logger.LogDebug("AnalyzerService.ScanPositions [TaskId: {TaskId}, TaskType:
{TaskType}]",
            command.TaskId, command.TaskType);

        switch (command.TaskType)
        {
            case "HeadHunter":
                var workers = _dataService.FindWorkersByCriteria(command.Criteries);

                return workers.Select(t => new AnalyzerResult
                {
                    TaskId = command.TaskId,
                    PositionId = t
                }).ToList();
            case "Worker":
                var vacancies = _dataService.FindVacanciesByCriteria(command.Criteries);

                return vacancies.Select(t => new AnalyzerResult
                {
                    TaskId = command.TaskId,
                    PositionId = t
                }).ToList();
        }

        throw new Exception($"Unknown TaskType {command.TaskType}");
    }
```

```
    public void SaveSearchPositionsResult(List<AnalyzerResult> positions)
    {
        _logger.LogDebug("AnalyzerService.SaveSearchPositionsResult [PositionsCount:
{Count}]",
            positions.Count);

        _dataService.SavePositionsResult(positions);
    }
}

using JobAssistant.BusinessLayer;
using JobAssistant.BusinessLayer.Analyzer;
using Microsoft.Extensions.Hosting;
using NLog;
using NLog.Extensions.Hosting;
using Quartz;

var logger = LogManager.Setup().LoadConfigurationFromFile().GetCurrentClassLogger();

logger.Info("Start Analyzer ...");

var builder = CreateHostBuilder(args);

builder.Build().Run();

static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .UseNLog()
        .ConfigureServices((hostContext, services) =>
        {
            services
                .AddQuartz(q =>
                {
                    q.SchedulerId = "Scheduler-Core";
                    q.UseMicrosoftDependencyInjectionJobFactory();
                    q.UseSimpleTypeLoader();
                    q.UseInMemoryStore();
                    q.UseDefaultThreadPool(tp =>
                    {
                        tp.MaxConcurrency = 10;
                    });
                    q.ScheduleJob<AnalyzerJob>(trigger => trigger
                        .WithIdentity("Combined Configuration Trigger")

.StartAt(DateBuilder.EvenSecondDate(DateTimeOffset.UtcNow.AddSeconds(7)))
                        .WithDailyTimeIntervalSchedule(x => x.WithInterval(20,
IntervalUnit.Second))
                        .WithDescription("my awesome trigger configured for a job with
single call")
                    );
                });

            services.AddProviderServices();

            services.AddQuartzHostedService(options =>
            {
                // when shutting down we want jobs to complete gracefully
                options.WaitForJobsToComplete = true;
                // when we need to init another IHostedServices first
                options.StartDelay = TimeSpan.FromSeconds(10);
            });
        });
using JobAssistant.BusinessLayer;
```

```
using JobAssistant.BusinessLayer.Parser;
using Microsoft.Extensions.Hosting;
using NLog;
using NLog.Extensions.Hosting;
using Quartz;

var logger = LogManager.Setup().LoadConfigurationFromFile().GetCurrentClassLogger();

logger.Info("Start Parser ...");

var builder = CreateHostBuilder(args);

builder.Build().Run();

static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .UseNLog()
    .ConfigureServices((hostContext, services) =>
    {
        services
            .AddQuartz(q =>
        {
            q.SchedulerId = "Scheduler-Core";
            q.UseMicrosoftDependencyInjectionJobFactory();
            q.UseSimpleTypeLoader();
            q.UseInMemoryStore();
            q.UseDefaultThreadPool(tp =>
            {
                tp.MaxConcurrency = 10;
            });
            q.ScheduleJob<ParserSchedulerJob>(trigger => trigger
                .WithIdentity("Combined Configuration Trigger")
                .StartAt(DateBuilder.EvenSecondDate(DateTimeOffset.UtcNow.AddSeconds(7)))
                .WithDailyTimeIntervalSchedule(x => x.WithInterval(5,
IntervalUnit.Second))
                .WithDescription("my awesome trigger configured for a job with single
call")
            );
        });

        services.AddProviderServices();

        services.AddQuartzHostedService(options =>
        {
            // when shutting down we want jobs to complete gracefully
            options.WaitForJobsToComplete = true;
            // when we need to init another IHostedServices first
            options.StartDelay = TimeSpan.FromSeconds(10);
        });
    });



using JobAssistant.BusinessLayer;
using JobAssistant.BusinessLayer.NotificationSender;
using Microsoft.Extensions.Hosting;
using NLog;
using NLog.Extensions.Hosting;
using Quartz;

var logger = LogManager.Setup().LoadConfigurationFromFile().GetCurrentClassLogger();

logger.Info("Start NotificationParser ...");
```

```
var builder = CreateHostBuilder(args);

builder.Build().Run();

static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .UseNLog()
        .ConfigureServices((hostContext, services) =>
        {
            services
                .AddQuartz(q =>
                {
                    q.SchedulerId = "Scheduler-Core";
                    q.UseMicrosoftDependencyInjectionJobFactory();
                    q.UseSimpleTypeLoader();
                    q.UseInMemoryStore();
                    q.UseDefaultThreadPool(tp =>
                    {
                        tp.MaxConcurrency = 10;
                    });
                    q.ScheduleJob<NotificationSenderJob>(trigger => trigger
                        .WithIdentity("Combined Configuration Trigger")

.StartAt(DateBuilder.EvenSecondDate(DateTimeOffset.UtcNow.AddSeconds(7)))
                        .WithDailyTimeIntervalSchedule(x => x.WithInterval(10,
IntervalUnit.Second))
                        .WithDescription("my awesome trigger configured for a job with
single call")
                    );
                });

            services.AddProviderServices();

            services.AddQuartzHostedService(options =>
            {
                // when shutting down we want jobs to complete gracefully
                options.WaitForJobsToComplete = true;
                // when we need to init another IHostedServices first
                options.StartDelay = TimeSpan.FromSeconds(10);
            });
        });


using JobAssistant.BusinessLayer;
using JobAssistant.WebPortal;
using NLog;
using NLog.Web;

var logger = LogManager.Setup().LoadConfigurationFromFile().GetCurrentClassLogger();

logger.Debug("Init JotAssistant.WebPortal ...");

try
{
    var builder = WebApplication.CreateBuilder(args);

    builder.Services.AddControllers();

    builder.Services.AddEndpointsApiExplorer();

    builder.Services.AddMvc();

    builder.Logging.ClearProviders();
```

```
    builder.Logging.AddNLogWeb();

    builder.Services.AddProviderServices();

    var app = builder.Build();

    app.UseMiddleware<ExceptionHandlingMiddleware>();

    app.UseStaticFiles();

    app.UseRouting();

    app.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");

    app.Run();
}
catch (Exception ex)
{
    logger.Error(ex, "Unhandled exception has occured {Message}", ex.Message);

    throw;
}
finally
{
    LogManager.Shutdown();
}
```

```
using System.Diagnostics.CodeAnalysis;
using System.Net;
using Newtonsoft.Json;

namespace JobAssistant.WebPortal;

/// <summary>
/// Global exception handler
/// </summary>
public class ExceptionHandlingMiddleware
{
    /// <summary>
    /// The next
    /// </summary>
    private readonly RequestDelegate _next;

    /// <summary>
    /// The logger
    /// </summary>
    private readonly ILogger<ExceptionHandlingMiddleware> _logger;

    /// <summary>
    /// Construct global exception handler
    /// </summary>
    /// <param name="next"></param>
    /// <param name="logger"></param>
    public ExceptionHandlingMiddleware(RequestDelegate next,
ILogger<ExceptionHandlingMiddleware> logger)
    {
        _next = next;
        _logger = logger;
    }

    /// <summary>
```

```
    /// Execute next handler with exception handling
    /// </summary>
    /// <param name="httpContext"></param>
    public async Task InvokeAsync(HttpContext httpContext)
    {
        try
        {
            await _next(httpContext);
        }
        catch (Exception ex)
        {
            await HandleExceptionAsync(httpContext, ex);
        }
    }

    /// <summary>
    /// Handles the exception using the specified context
    /// </summary>
    /// <param name="context">The context</param>
    /// <param name="exception">The exception</param>
    private async Task HandleExceptionAsync(HttpContext context, Exception exception)
    {
        context.Response.ContentType = "application/json";

        var response = context.Response;

        var errorResponse = new ErrorResponse
        {
            Success = false
        };

        switch (exception)
        {
            case ApplicationException ex:
                if (ex.Message.Contains("Invalid Token"))
                {
                    response.StatusCode = (int) HttpStatusCode.Forbidden;

                    errorResponse.Message = ex.Message;

                    break;
                }

                response.StatusCode = (int) HttpStatusCode.BadRequest;

                errorResponse.Message = ex.Message;

                break;
            default:
                response.StatusCode = (int) HttpStatusCode.InternalServerError;

                errorResponse.Message = "Internal server error";

                break;
        }

        _logger.LogError("Global exception handler: {ExceptionMessage}, {StackTrace}",
ErrorMessage(exception),
            exception.StackTrace);

        var result = JsonConvert.SerializeObject(errorResponse);

        await context.Response.WriteAsync(result);
    }
```

```
    /// <summary>
    /// Gets the error message using the specified exception
    /// </summary>
    /// <param name="exception">The exception</param>
    /// <returns>The message</returns>
    private static string ErrorMessage(Exception exception)
    {
        var message = exception.Message;

        if (!string.IsNullOrEmpty(exception.InnerException?.Message))
        {
            message += ErrorMessage(exception.InnerException);
        }

        return message;
    }
}

/// <summary>
/// The error response class
/// </summary>
[SuppressMessage("ReSharper", "UnusedAutoPropertyAccessor.Global")]
public class ErrorResponse
{
    /// <summary>
    /// Gets or sets the value of the success
    /// </summary>
    public bool Success { get; set; }

    /// <summary>
    /// Gets or sets the value of the message
    /// </summary>
    public string? Message { get; set; }
}


using System.Text;
using JobAssistant.BusinessLayer.Portal;
using JobAssistant.WebPortal.Models;
using Microsoft.AspNetCore.Mvc;

namespace JobAssistant.WebPortal.Controllers;

public class HomeController : Controller
{
    private readonly IUsersService _usersService;

    private readonly ITasksService _tasksService;

    private readonly IAnalyzedTaskService _analyzedTaskService;

    private readonly ILogger<HomeController> _logger;

    public HomeController(IUsersService usersService, ILogger<HomeController> logger,
ITasksService tasksService,
        IAnalyzedTaskService analyzedTaskService)
    {
        _usersService = usersService;
        _logger = logger;
        _tasksService = tasksService;
        _analyzedTaskService = analyzedTaskService;
    }

    public ActionResult Index()
```

```
    {
        return View();
    }

    public ActionResult HeadHunters()
    {
        return View();
    }

    public ActionResult Workers()
    {
        return View();
    }

    [HttpGet]
    public async Task<JsonResult> Authenticate(string credentials)
    {
        try
        {
            credentials = Base64Decode(credentials);

            if (!credentials.Contains(':'))
            {
                return Json(new {Success = false, Message = "Wrong login or password"});
            }

            var login = credentials.Substring(0, credentials.IndexOf(":",
StringComparison.Ordinal));

            var password = credentials.Substring(login.Length + 1, credentials.Length -
login.Length - 1);

            var user = await _usersService.Login(login, password);

            return user == null
                ? Json(new {Success = false, Message = "Access denied"})
                : Json(new {Success = true, UserType = user.Type, user.FullName});
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Authentication exception");

            return await Task.FromResult(Json(new {Success = false, Message = "Internal
error"}));
        }
    }

    [HttpGet]
    public async Task<JsonResult> GetTasks(string userLogin)
    {
        try
        {
            var result = new List<TaskDetailsDto>();

            foreach (var task in await _tasksService.GetTasksByUser(userLogin))
            {
                var criteries = task.Criteries.Aggregate((a, b) => $"{a} {b}");

                criteries = criteries.Length != 0
                    ? criteries[..^1]
                    : string.Empty;

                result.Add(new TaskDetailsDto(task.TaskId, task.Name, criteries,
task.PositionsFound,
```

```
                    task.LastAnalyzed?.ToShortTimeString() ?? string.Empty));
            }

            return Json(new {Success = true, Data = result});
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "GetTasks exception [UserLogin: {UserLogin}",
userLogin);

            return await Task.FromResult(Json(new {Success = false, Message = "Internal
error"}));
        }
    }

    [HttpGet]
    public async Task<JsonResult> GetPositions(int taskId)
    {
        try
        {
            return Json(new {Success = true, Data = await
_analyzedTaskService.GetTaskPositions(taskId)});
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "GetPositions exception [TaskId: {TaskId}", taskId);

            return await Task.FromResult(Json(new {Success = false, Message = "Internal
error"}));
        }
    }

    private static string Base64Decode(string value)
    {
        var data = Convert.FromBase64String(value);

        return Encoding.UTF8.GetString(data);
    }
}
```