КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття ступеня магістра
за спеціальністю 121 Інженерія Програмного Забезпечення
на тему:
Розподілена система автоматичної агрегації вакансій

Виконав:
студент 2-го курсу
Дмитро БРАНІЦЬКИЙ _____
(підпис)

Науковий керівник:
доцент, кандидат фізико-математичних наук
Олексій ЧЕНЦОВ _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
«12» травня 2021 р.,
протокол № 12

Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ - 2021

РЕФЕРАТ

Обсяг роботи 62 сторінок, 13 ілюстрацій, 17 джерел посилань.

СКРЕПІНГ, АВТОМАТИЧНА АГРЕГАЦІЯ, АГРЕГАЦІЯ ІНФОРМАЦІЇ, АГРЕГАЦІЯ ВАКАНСІЙ, РОЗПОДІЛЕНА СИСТЕМА

Об’єктом роботи є задача розподіленої автоматичної агрегації інформації з мережі Інтернет, а саме - агрегація вакансій з відповідних ресурсів.

Предметом роботи є побудова розподіленого алгоритму автоматичної агрегації вакансій, та безпосередньо побудова розподіленої системи автоматичної агрегації вакансій. Система повинна бути розподіленою та спроектована таким чином, щоб мати можливість масштабування розрахункових потужностей, а також можливість розширення списку підтримуваних джерел інформації, без необхідності внесення змін в існуючу архітектуру системи.

Метою роботи є створення програмного забезпечення, що реалізує відповідну розподілену систему автоматичної агрегації вакансій, яка може легко масштабуватися за рахунок своєї розподільності. Система повинна автоматично відслідковувати появу нових вакансій, агрегувати їх, та відслідковувати зміну статусу вакансії з часом.

Інструменти розроблення: середовище розробки PyCharm 2021.1 (Professional Edition), мова програмування Python версії 3.9, система версіювання файлів GIT.

Результати роботи: створено програмний продукт у вигляді розподіленої системи автоматичної агрегації вакансій з таких ресурсів: djinni.co та rabota.ua. Система побудована на розподіленій архітектурі, з використанням **Celery** - в якості черги розподілених задач, **RabbitMQ** - в якості брокеру повідомлень для Celery, а також веб-фреймворк **Django**, для використання Django ORM при роботі з базою даних (в тому числі і для міграцій), а також можливості в подальшому розширити додаток веб-інтерфейсом. Також використовується технологія віртуалізації **Docker** - для можливості запуску системи незалежно від операційної системи сервера, а також для можливості легкого масштабування системи і в

подальшому використання засобів аркестрації контейнерів (наприклад Kubernetes). Разом з Docker, для запуску системи використовується **docker compose**. Після запуску та деякого часу роботи системи - в базу даних починає надходити агрегована інформація з детальних сторінок вакансій.

Значимість роботи: даний продукт розрахований в основному на агрегацію вакансій в сфері Інформаційних технологій, при цьому в останні роки (в останні місяці це стало значно помітніше) кількість вакансій та кількість платформ з розміщення вакансій в сфері Інформаційних технологій - стрімко зростає. Це в свою чергу призводить до зменшення ефективності кожної з платформ, та збільшує ймовірність того, що потенційно цікава для фахівця вакансія - залишиться непоміченою фахівцем, оскільки фахівцю важко відслідковувати всі платформи з розміщення вакансій. Для запобігання цієї ситуації деякі компанії витрачають час на розміщення одних й тих самих вакансій одночасно на декількох платформах.

Система, створена в результаті виконання цієї роботи, надає можливість агрегувати IT вакансії з різних платформ в одному місці. Це в свою чергу надає можливість користувачам, зацікавленим у вакансіях в сфері Інформаційних технологій, використовувати агреговану інформацію для відслідковування / пошуку / фільтрації цікавих їм вакансій та даних.

До того ж, архітектура даної системи дозволяє легко використовувати її для розподіленої агрегації схожих інформаційних ресурсів (блогів, платформ з оголошеннями).

Пропозиції щодо розвитку об’єкта розроблення: неодмінно можна розширювати список джерел вакансій. Також можна вдосконалити веб інтерфейс додавши можливість зручного перегляду, фільтрації та пошуку по агрегованій інформації. Зручний веб інтерфейс дасть змогу зробити публічним доступ до агрегованої інформації.

Серед наступних етапів розвитку системи - можна додати використання таких технологій як **Splash**, що дозволить додавати в якості джерел інформації сайти, що потребують JavaScript для відображення інформації на стороні клієнта (оскільки зараз для витягу інформації використовуються лише HTTP запити, без

можливості інтерактивної взаємодії зі сторінками та без можливості відображення інформації на сайтах, що використовують для цього JavaScript). Також за необхідності це довзолить реалізувати механізми вирішення Captcha різних типів, якими деякі веб-ресурси можуть захищатись від скрепінгу.

До того ж, в майбутньому можна додати використання proxy, задля запобігання перевищення ліміту запитів з одної IP адреси, при агрегації великої кількості вакансій з одного ресурсу. А також це дозволить приховати IP адреси Celery worker-ів системи, що в свою чергу унеможливлює прямі атаки на систему агрегації вакансій.

Також в майбутньому можна додати агрегацію інформації про компанії-роботодавців, і, відповідно, відображення вакансій, згрупувавши їх в залежності від роботодавця.

Висновки: в результаті виконання роботи були проаналізовані різні платформи з розміщення вакансій та створена картина типової платформи з розміщення вакансій. Були сформовані вимоги до розподіленого алгоритму автоматичної агрегації вакансій та оновлення статусу вакансій в часі. Після цього були створені відповідні алгоритми. Потім - були сформовані вимоги до розподіленої системи автоматичної агрегації, що реалізовували б ці алгоритми, та на основі сучасних технологій була побудована архітектура, що задовільняє сформовані вимоги до системи. Після цього, на основі побудованої архітектури була реалізована відповідна розподілена система автоматичної агрегації вакансій.

Отримане програмне забезпечення дозволяє розподілено автоматично агрегувати вакансії з декількох джерел та оновлювати статус вакансій в часі (перевіряти чи не закрилась вакансія). Отримана система є розподіленою та масштабованою, що дозволяє без складнощів працювати з великою кількістю джерел та даних на цих джерелах. Система може бути розширена новими платформами з розміщення вакансій в якості джерела інформації, без змін існуючої архітектури.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

IT - Інформаційні технології;

ОС - Операційна система;

app - Application (застосунок);

ORM - Object–relational mapping (Об’єктно-реляційне відображення);

JS - JavaScript (мова програмування).

ВСТУП

Оцінка сучасного стану об’єкта розробки. На даний час питання агрегації інформації з мережі Інтернет є доволі актуальним. Ця сфера має багато напрямів, в більшості своїй пов’язаних з подальшим аналізом даних: задля зменшення ризиків безпеки, маркетингу, соціальних передбачень, відображення інформації в єдиному ресурсі. Оскільки ця сфера актуальна - розроблюються та підтримуються різні технології, що якоюсь мірою пов’язані з подібними задачами агрегації інформації з мережі Інтернет. Хоча дійсно існують фреймворки для подібних задач (наприклад, бібліотека Scrapy для Python), все ж готових рішень на кожен випадок - немає. Тому доводиться комбінувати різні технології заради досягнення мети, в залежності від кожної з задач. Що логічно, оскільки подібні задачі завжди є доволі специфічними та різноманітними, а комбінування технологій, в свою чергу, надає гнучкості для створення систем під самі різні задачі. Якщо говорити про розподілені системи агрегації інформації - немає готових рішень, що можуть бути відразу застосовані. Відповідно й немає готових рішень для системи розподіленої агрегації вакансій. Але сучасний розвиток Інформаційних технологій все ж дозволяє реалізовувати подібні системи, знову ж таки, комбінуванням технологій.

Актуальність роботи та підстави для її виконання. На даний час існує багато платформ для розміщення вакансій. Але коли нові люди намагаються щось змінити в цій сфері, вони створюють ще одну таку ж платформу, й намагаються конкурувати з усіма іншими, що в цілому зменшує ефективність кожної з платформ. Варто відмітити, що такі платформи для розміщення вакансій часто називають себе “агрегаторами вакансій”, але вони “агрегують” вакансії лише в тому сенсі, що вакансії публікуються на цих сайтах, й по суті такі проекти не агрегують вакансії з інших платформ.

Що стосується саме агрегації вакансій в розрізі процесу агрегації (а не публікації вакансій на ресурсі) - існував український проект, в дечому схожий до розроблюваної системи - Hotwork. В основі цього проекту була покладена схожа

до моєї роботи ціль - агрегація вакансій з різних платформ, що розміщують вакансії. З відмінностей - цей проект агрегував вакансії різних сфер, а не виключно IT. Але у 2018 році проект Hotwork викупила українська компанія Jooble за суму в 30000$.

В свою чергу, існують такі проекти як Jooble, що на відміну Hotwork, все ж не агрегують вакансії, а, знову ж таки, являють собою чергову платформу з розміщення вакансій. До того ж, на платформі розміщуються вакансії різних сфер, а не виключно IT. Ще одна відмінність Jooble - проект націлений на багато ринків, але кожен - в середині певної країни (ua.jooble.com - для українського ринку, uk.joobe.com - для англійського ринку і т.д.), що створює нову проблему у вигляді необхідності конкуренції з іншими платформами в середині кожної з країн, що є доволі складною задачею.

При цьому на даний момент немає проектів, що агрегують вакансії з різних місцевих платформ, націлені на вакансії в сфері IT, та при цьому є інтернаціональними та міжнародними в тому сенсі, що намагаються звести фахівців та роботодавців з усього світу.

Мета й завдання роботи. Мета: дослідити платформи з розміщення вакансій, за для виявлення спільних рис; сформувати картину типової платформи з розміщення вакансій; створити розподілений алгоритм з агрегації інформації з подібних платформ; спроектувати архітектуру розподіленої системи з агрегації вакансій, що реалізує сформовані алгоритми та використовує наявні сучасні технології; та створити відповідне програмне забезпечення, що реалізує спроектовану архітектуру. Результуюча система повинна бути достатньо гнучкою, щоб мати змогу розширювати список джерел інформації новими веб-ресурсами без змін існуючої архітектури.

Об’єкт, методи й засоби розроблення. Об’єктом розподіленої системи автоматичної агрегації вакансій є саме проектування реалізації подібної розподіленої системи автоматичної агрегації даних на основі сучасних технологій, оскільки з швидкісним зростанням джерел інформації в мережі Інтернет - ця задача залишається актуальною для різних цілей.

Проектуванню та реалізації системи передував аналіз платформ з розміщення вакансій, виявлення спільних рис, а також підбір технологій, що дозволяють створювати розподілені системи та автоматизувати процеси.

Можливі сфери застосування. Першочергова сфера застосування - агрегація вакансій в сфері IT та пошук, аналіз, фільтрація по агрегованій інформації. Надалі можна надати певний інтерфейс до цих даних. Інші можливості утвореної системи - незначна модифікація та використання механізму розподіленої автоматичної агрегації - для агрегування інформації з інших схожих за форматом ресурсів (сайтів новин, блогів, тощо).

РОЗДІЛ 1 НЕОБХІДНІСТЬ АНАЛІЗУ ПЛАТФОРМ З РОЗМІЩЕННЯ ВАКАНСІЙ

При вирішенні задачі побудови розподіленої системи з автоматичної агрегації вакансій - необхідно проаналізувати та підібрати ресурси, з яких ми будемо агрегувати дані. До того ж, що б агрегувати дані, необхідно розуміти в якому форматі ці дані відображаються на ресурсах, та що саме з цих даних може бути корисно агрегувати. Інакше подальше проектування системи просто неможливо, оскільки без аналізу джерел інформації неможливо утворити алгоритм з агрегації цієї інформації, моделі бази даних, процес автоматизації агрегації, та інше.

1.1 Необхідність аналізу декількох різнотипних джерел

Після детального аналізу та опису структури веб-ресурсу (платформи з розміщення вакансій) та розміщених на даному веб-ресурсі даних - слід врахувати, що навіть якщо на певному з ресурсів присутні додаткові дані, що можуть бути корисними й що є доступними саме на цьому ресурсі - можуть бути відсутні на інших ресурсах.

Якщо цього не врахувати та реалізувати систему базуючись на першому ж веб-ресурсі в якості джерела інформації - подальше додавання кожного нового джерела буде потребувати значних змін системи, її архітектури, моделей бази даних та інше - що є протилежністю до нашої мети, оскільки задача полягає у створенні гнучкої системи, список джерел якої може бути легко розширений новими веб-ресурсами. Саме тому насамперед необхідно проаналізувати **декілька** ресурсів, бажано різнотипних.

1.2 Точки уваги при аналізі джерел інформації

При аналізі кожного з ресурсів необхідно звернути увагу на такі речі як:

- структура ресурса;
- наявність сортування списку вакансій за датою опублікування;
- наявність фільтрації;
- наявність тегів або інших рис відмінності / категоризації вакансій;
- наявність пагінації;
- наявність детальних сторінок вакансій;
- необхідність виконання JavaScript для отримання часткової або повної інформації;
- структурні блоки в описі вакансій та можливість їх виокремлення;
- всі наявні дані, наявні для кожної з вакансій.

Після аналізу декількох ресурсів, необхідно виділити наявні спільні риси між різними ресурсами, та спільні риси між даними на різних ресурсах. Це необхідно для того, щоб проектувати систему з урахуванням сами спільних загальних рис, щоб в майбутньому можна б розширити список джерел ресурсами, що мають подібні загальні риси, без необхідності змін самої системи.

Варто ще раз підкреслити необхідність виділення таких рис, що не є специфічними, а є якомога загальними. Це надасть змогу використовувати в якості джерел - навіть структурно нетипові платформи з розміщення вакансій.

1.3 Підбір джерел інформації для подальшого аналізу

При виборі джерел інформації (платформ з розміщення вакансій) я насамперед керувався наступними критеріями:

- це повинна бути популярна платформа, що розміщує значну кількість вакансій та має значну кількість користувачів;
- платформа має бути українською;

- платформа має розміщувати виключно вакансії в сфері Інформаційних технологій, або ж мати чітко відокремлений розділ для IT вакансій.

Отже, для подальшого аналізу було обрано наступні платформи:

- djinni.co
- jobs.dou.ua
- it.rabota.ua

РОЗДІЛ 2 АНАЛІЗ ПЛАТФОРМ З РОЗМІЩЕННЯ ВАКАНСІЙ

Спочатку проаналізуємо та вивчимо структуру кожного з обраних ресурсів, а далі перейдемо до створення картини типової платформи з розміщення вакансій, та до визначення даних, що доступні на кожній з платформ, для подальшої побудови універсальної системи, яка буде достатньо загальною, щоб агрегувати дані з різних платформ та мати можливість розширення списку джерел.

2.1 Аналіз обраних веб-ресурсів з розміщення вакансій

В першому розділі були описані деталі, на які необхідно звернути увагу при аналізі ресурсів. Перейдемо безпосередньо до аналізу обраних ресурсів, враховуючи необхідні “точки уваги”.

2.1.1 Аналіз djinni.co

Перейшовши на сайт проекту djinni.co, та відкривши розділ з вакансіями - https://djinni.co/jobs/, можна побачити доволі типову і в той самий час доволі загальну структуру майже будь яких інформаційних ресурсів, що мають публікації / оголошення (від деяких онлайн платформ для обміну товарами та послугами, продовжуючи форумами та платформами з розміщення вакансій). Загальний вигляд платформи наступний: список вакансій, відсортований в порядку публікації (нові - зверху), кожне оголошення має посилання на детальну сторінку.

На головній сторінці з вакансіями відразу видно ті деталі, на які варто звернути увагу:

1) На платформі доступний пошук по вакансіям.
2) Також доступна система фільтрів по наступним критеріям:
   - технології, що задіяні в вакансії;
   - місто вакансії (або ж віддалено);

- необхідний досвід роботи фахівця;
- рівень заробітної плати;
- часткова зайнятість.

3) Доступна пагінація сторінок на наступну та попередню сторінку. Хоча в дійсності, при спробі перейти таким чином до останньої сторінки, починаючи з близько 650-тої сторінки, при спробі перейти на наступну сторінку - платформа починає повертати не чергову сторінку, а останню доступну (поточну), хоча при цьому кнопка пагінації для переходу на наступну сторінку залишається доступною, і загальна кількість “переглянутих” вакансій - не співпадає з загальних числом вакансій, що вказане на першій сторінці (якщо перемножити кількість вакансій на одній сторінці з кількістю доступних сторінок - отримаємо кількість “переглянутих” вакансій, що ми зустріли на всіх минулих сторінках).

4) Немає необхідності у виконанні JavaScript коду задля отримання будь якої інформації (основної чи додаткової).

5) Список вакансій відсортований за датою публікації вакансій.

Тепер виділимо, які саме дані про кожну з вакансій можна отримати з головної сторінки. При перегляді списку вакансій, для кожної вакансії є наступні дані:

- назва вакансії;
- короткий опис вакансії;
- посилання на детальну сторінку вакансії (її розглянемо нижче);
- дата публікації вакансії;
- межі заробітної плати;
- необхідний досвід фахівця в роках;
- необхідний рівень англійської мови;
- місто, в якому знаходиться посада;
- дані про автора публікації:
  - ім’я та фото;
  - посада;

- назва компанії;
- посилання на сторінку автора, що в свою чергу містить такі дані:

а) дату реєстрації автора на платформі;

б) список поточно відкритих автором вакансій;

в) список відкритих вакансій компанії, що розміщені на платформі (вне залежності від автора публікації);

г) розділ з описом компанії та посиланням на сайт компанії, а також на сторінку компанії на сайті dou.ua, яка в свою чергу містить опис компанії, приблизну кількість працівників, відкриті вакансії компанії, що розміщені на платформі dou.ua, фото офісу компанії, відгуки працівників про компанію, та посилання на сайт самої компанії.

Наразі ми описуємо не тільки ті дані, що можуть бути нам цікаві, а всі доступні, щоб надалі можна було проаналізувати, порівняти, та узагальнити доступні дані серед різних ресурсів.

Розглянемо детальну сторінку вакансії. На детальній сторінці можна побачити наступні дані:

- назва вакансії;
- короткий опис вакансії (той самий, що його видно на головній сторінці зі списком вакансій);
- розширений опис вакансії (в якому зазвичай описують проект, команду, компанію, клієнта, вимоги до фахівця, використовувані технології, переваги роботи в даній компанії / на цьому проекті / на даній посаді);
- дата публікації вакансії;
- межі заробітної плати;
- необхідний досвід фахівця в роках;
- необхідний рівень англійської мови;
- місто, в якому знаходиться посада;
- ім’я, фото, посада, та компанія автора публікації;

- розділ з описом про компанію та посиланням на сайт компанії.

Це всі наявні на даному ресурсі дані. Пізніше співставимо це з наявними на інших ресурсах даними та виокремимо загальну структуру подібних ресурсів задля формування алгоритму агрегації, а також виділимо спільні розділи, які будемо агрегувати.

2.1.2 Аналіз jobs.dou.ua

Перейдемо до аналізу наступного обраного ресурсу. Перейшовши на головну сторінку розділу з вакансіями - https://jobs.dou.ua, ми бачимо підрозділи, що розділяють вакансії по технологіям, і декілька вакансій в кожному з підрозділів. Але нижче все ж є відсортований список всіх вакансій.

Ключові деталі сторінки:

1) Доступний пошук по вакансіям.

2) Доступна система фільтрів по наступним критеріям:

- напрям діяльності (мова програмування, маркетинг, тощо);
- місто вакансії (або ж віддалено);
- необхідний досвід роботи фахівця.

3) Пагінація сторінок відсутня. Переглянути наступну порцію вакансій можна натиснувши кнопку “Більше вакансій”, яка динамічно підгружає на цю ж сторінку додатково 40 вакансій. Це призводить до 4го пункту.

4) Є необхідність у виконанні JavaScript для отримання списку вакансій, оскільки динамічне завантаження наступної порції вакансій при натисканні кнопки “Більше вакансій” - імплементовано за рахунок роботи JavaScript в браузері. Таким чином, якщо програмна система буде потребувати відтворювати такі дії - необхідно завантажувати сторінку засобами, що мають можливість обробляти JavaScript сторінки.

5) Список вакансій відсортований за датою публікації вакансій.

На головній сторінці зі списком вакансій, для кожної з вакансій можна побачити наступні дані:

- назва вакансії;
- короткий опис вакансії (в якому зазвичай описують проекту або компанію);
- посилання на детальну сторінку вакансії;
- межі заробітної плати;
- місто, в якому знаходиться посада (може бути декілька, в тому числі й віддалено);
- назва компанії з посиланням на сторінку компанії на порталі dou.ua.

Розглянемо детальну сторінку вакансії. На детальній сторінці вакансії доступні наступні дані:

- назва вакансії;
- дата публікації оголошення;
- місто, в якому знаходиться посада (може бути декілька, в тому числі й віддалено);
- короткий опис компанії з посиланням на всі вакансії компанії та на сторінку компанії на порталі dou.ua, що містить серед іншого містить опис компанії, інформацію про приблизну кількість працівників, фото офісу, відгуки працівників про компанію, та посилання на сайт компанії;
- розширений опис вакансії, що не завжди, але часто поділений на наступні розділи:
    - необхідні навички;
    - “буде плюсом”;
    - яку компенсацію компанія пропонує (умови роботи, заробітню плату);
    - обов’язки, що передбачає позиція.
- короткий опис вакансії (в якому зазвичай описують проекту або компанію).

Загалом це вся інформація, щодо вакансії, яка доступна на даному ресурсі. Перейдемо до розгляду наступного ресурсу - it.rabota.ua.

2.1.3 Аналіз it.rabota.ua

Варто відразу відмітити, що сайт rabota.ua дозволяє розміщувати будь-які вакансії, а не тільки вакансії в сфері Інформаційних технологій. Але ж ми будемо розглядати лише ту частину платформи, яка чітко призначена лише для вакансій в IT сфері.

Перейшовши на головну сторінку IT розділу сайту rabota.ua - https://it.rabota.ua, ми можемо бачити декілька вакансій з відповідними посиланнями та посиланням на список всіх вакансій. Також на головній сторінці можна побачити декілька провідних компаній з посиланнями на вакансії від кожної з цих компаній, та деякі мови програмування та напрями з посиланнями на список вакансій, де фігурують відповідна мова програмування / напрям. Перейдемо на сторінку зі списком всіх вакансій.

Ключові деталі сторінки:

1) Доступний пошук по вакансіям.

2) Доступна система фільтрів по наступним критеріям:

- напрям діяльності та технології (мова програмування, маркетинг, тощо);
- вид зайнятості (часткова, повна, віддалена, проектна, стажування);
- місто вакансії (або ж віддалено);
- рівень заробітної плати;
- рівень посади (молодший спеціаліст, топ менеджер, та інші).

3) Присутня пагінація сторінок з можливістю переходу на декілька попередніх сторінок, декілька наступних сторінок, та першу і останню сторінки.

4) Немає необхідності у виконанні JavaScript для отримання основної чи будь якої додаткової інформації.

5) Список вакансій відсортований за датою публікації вакансій.

Розглянемо, які дані про вакансії можна отримати зі сторінки списку вакансій. Для кожної з вакансій можна отримати такі дані:

- назва вакансії;
- посилання на детальну сторінку вакансії;
- короткий опис вакансії (що може містити опис проекту або компанії);
- рівень заробітної плати;
- назва та логотип компанії;
- дата публікації;
- набір тегів, що відносяться до вакансії (“Регулярний перегляд зарплат”, “Медичне страхування”, тощо).

Розглянемо детальну сторінку вакансії. На детальній сторінці вакансії можна побачити наступні дані:

- назва вакансії;
- дата публікації;
- рівень заробітної плати;
- назва компанії та посилання на сторінку компанії на сайті rabota.ua (що містить список вакансій компанії, опублікованих на цій платформі);
- адреса офісу (або місто);
- особливості вакансії, в серед іншого які входять наступні деталі:
    а) напрями діяльності та технології;
    б) необхідний рівень навичок фахівця;
    в) необхідне знання іноземних мов;
    г) вид зайнятості;

- детальний опис вакансії, що може містити опис компанії, проекту, команди, необхідні навички, пропонуванні умови праці та матеріальна компенсація.

Загалом, це всі дані про вакансії, що доступні на даному ресурсі.

2.2 Типова платформа з розміщення вакансій

Тепер проаналізуємо зібрану інформацію з метою пошуку спільних загальних характеристик: структури веб-ресурсу та даних про вакансію.

2.2.1 Структура веб-ресурсу

Після аналізу обраних ресурсів, ми можемо описати типову структуру подібних платформ з розміщення вакансій.

Як ми змогли побачити, в основі кожного з проаналізованих ресурсів покладена наступна типова структура: список вакансій з посиланнями на окремі сторінки вакансій, що містять детальну інформацію щодо відповідної вакансії. До того ж, списки відсортовані за датою публікації (що грає важливу роль в оптимізації, але про це буде в розділах про алгоритм агрегації). І ця структура є доволі загальною для всіх подібних ресурсів - що є “дошками оголошень”. Оскільки основна ціль подібні ресурсів - зручне та ергономічне відображення якомога більшої кількості елементів з коротким описом (щоб надати можливість користувачеві переглянути список, та зацікавити його певним з елементів цього списку), а після - надати більш детальну інформацію на розгорнутій сторінці. При чому, сортування оголошень по типу “нові оголошення - зверху” також є логічним для структури подібних ресурсів. Тому подібна структура використовується як на подібних платформах, так і при реалізації інтернет магазинів, сайтів новин, поштових скриньок, форумів, та іншого.

Тому таку структуру веб-ресурсу можна вважати достатньо загальною, щоб покласти її в основу алгоритму агрегації та зберегти при цьому можливість в

майбутньому розширювати список джерел інформації, оскільки нові джерела будуть так чи інакше мати подібну структуру.

2.2.2 Наявні дані вакансій та їх формат

Тепер виходячи з проаналізованих ресурсів - виділимо загальні дані, що відображені на всіх подібних платформах.

Отже з спільних даних наявні:

- назва вакансії;
- опис вакансії (вільного формату, з описом вимог, обов'язків, умов праці та компенсації);
- дата публікації;
- рівень заробітної плати;
- місто / віддалено;
- інформація про компанію;
- додаткова інформація різного вигляду (теги, розділ "про проект", та інше).

Подібно до типової структури - такий формат даних також є достатньо загальним, бо всі ці дані є спільною характеристикою всіх вакансій (дата публікації, умови праці, та інше). Ці дані при цьому є достатніми для повного опису вакансії. Тому на такому форматі даних цілком можна базувати моделі бази даних системи агрегації, оскільки при отриманій загальній моделі бази даних - система буде достатньо гнучкою та матиме можливість розширення списку джерел інформації новими веб-ресурсами.

2.3 Висновки

Проаналізувавши обрані платформи з розміщення вакансій ми змогли створити картину загальної типової платформи для розміщення вакансій, а саме - структуру типового ресурсу та формат даних про вакансію, що доступні на

типовому ресурсі. При цьому, і сформована структура ресурсу, і формат даних - є достатньо загальними, що й було нашою ціллю, оскільки це надає можливість реалізувати систему гнучкою, задля можливості розширення списку джерел інформації.

РОЗДІЛ 3 АЛГОРИТМ РОЗПОДІЛЕНОЇ АВТОМАТИЧНОЇ АГРЕГАЦІЇ

3.1 Вимоги до алгоритму розподіленої автоматичної агрегації

Тепер необхідно створити алгоритм агрегації інформації про вакансії, враховуючи отриману картину типової платформи з розміщення вакансій.

Схематично зображену структуру типової платформи з розміщення вакансій можна переглянути в **Додатку А**. Від цієї структури будемо відштовхуватись як при висуванні вимог до алгоритму, так і безпосередньо при створенні алгоритму. Спочатку необхідно сформувати вимоги до алгоритму.

Для того, щоб в кінцевому рахунку ми отримали розподілену систему, необхідно щоб алгоритм це враховував, тому перша вимога:

1) Агрегація інформації повинна бути розподіленою.

Розкриємо цей пункт трохи ширше обто мати можливість виконуватись не лише багатопотоково чи асинхронно, а й саме незалежно / розподілено. Мається на увазі, що процеси агрегації різних даних могли запускатися незалежно один від одного. Це повинно надати можливість різним процесам одночасно працювати навіть на різних машинах, таким чином заклавши можливість подальшого масштабування системи. [1]

Інша ключова ціль - зібрати детальний опис вакансії. Звідси випливає друга вимогу до алгоритму:

2) Алгоритм повинен в тому числі проходити та агрегувати детальні сторінки вакансій, а саме наступну інформацію:

- назва вакансії та посилання на вакансію на сайті ресурсу;
- назва компанії та посилання на сторінку компанії на сайті ресурсу;
- дата публікації;
- місто;

- детальний опис вакансії;

- додаткову інформацію, якщо вона є (теги, “про проект”, та інше, в залежності від ресурсу).

Варто відмітити що задля збереження гнучкості системи, наступні поля є опціональними:

- посилання на сайт компанії на ресурсі;

- додаткова інформація вакансії.

В якості інших вимог висунемо наступне:

3) Алгоритм повинен виявляти, коли попередньо агреговані вакансії закриваються / видаляються.

4) Алгоритм повинен не лише разово зібрати / агрегувати поточно наявні вакансії, а й регулярно виявляти появу нових вакансій та агрегувати їх.

5) Алгоритм повинен бути оптимальним. Мається на увазі, щоб система не витрачала час на запити для отримання інформації, що вже була раніше агрегована. Як ми побачимо далі, наявність сортування вакансій за датою публікації - нам в цьому допоможе. Це необхідно як для запобігання блокування ір адресів системи при перевищенні ліміту одночасних запитів в одну секунду, та прибере не потрібне навантаження як на самі ресурси, так і на розрахункові потужності самої системи агрегації.

Також, варто відмітити, що немає вимоги визначення змін в уже опублікованих вакансіях. За “дано” приймемо те, що вакансія після публікації не може бути змінена. Це, серед іншого, зумовлено тим, що не всі платформи можуть позначати оновлення вакансії, тому в найгіршому випадку для подібного відстежування оновлень вакансії - необхідно регулярно робити запити на кожну з вакансій та порівнювати повний текст оголошення з тим, що був раніше агрегований. Це є недоцільним у відношенні як до основної цілі системи, так і до інших вимог до алгоритму.

Тому це всі вимоги. Хоча вони і є доволі загальними, все ж їх дотримання забезпечить те, що алгоритм буде:

- розподіленим;
- оптимальним;
- агрегувати детальні дані по всім вакансіям;
- в тому числі й постійно агрегувати ново-опубліковані вакансії;
- визначати чи закрились ті оголошення, що вже були агреговані.

3.2 Побудова алгоритму розподіленої автоматичної агрегації

Отже, виходячи з наявної картини типової платформи з розміщення вакансій, та з вимог до алгоритму, можемо створити відповідний алгоритм агрегації інформації.

Спочатку створимо загальний простий алгоритм агрегації, а потім перейдемо до його вдосконалення та оптимізації, задля того, щоб алгоритм задовольняв всі попередньо сформовані вимоги.

3.2.1 Початковий алгоритм агрегації вакансій

Розглянемо типову структуру платформи з розміщення вакансій (див. **Додаток А**). В нас є головна сторінка, на якій міститься списком відсортованих вакансій з посиланнями на відповідні детальні сторінки цих вакансій. Щоб мати можливість зібрати всю необхідну інформації - необхідно заходити на відповідні детальні сторінки вакансій. Опишемо алгоритм агрегації всієї необхідної інформації, а саме - виявлення всіх детальних сторінок вакансій та обходу всіх цих сторінок:

1) завантажити “головну” сторінку платформи, що відображає відсортований список вакансій;

2) оберемо в роботу першу вакансію зі списку;

3) перейти на детальну сторінку вакансії;

4) агрегувати з поточної детальної сторінки всі необхідні дані;

5) повернутись на попередню сторінку;

6) оберемо в роботу наступну вакансію зі списку;

7) повторити пункти 3 - 6, доки не буде агрегована інформація для кожної з вакансій на поточній сторінці;

8) перейти на наступну сторінку списку вакансій;

9) повторити пункту 2 - 8, доки не буде агрегована інформація для кожної вакансій з кожної сторінки на платформі;

10) періодично повторювати пункти 1 - 9.

Sequence-діаграму цього алгоритму можна знайти в **Додатку Б**.

Ми сформували алгоритм агрегації всіх необхідних даних, і він задовольняє вимоги, що визначені пунктами 2 (агрегувати детальні сторінки) та 4 (регулярно агрегувати ново-опубліковані оголошення). Але все ж цей алгоритм не задовольняє інших вимог, а саме пункти 1 (розподільність алгоритму), 3 (виявлення події закриття агрегованих вакансій), та 5 (оптимальність)

3.2.2 Удосконалення алгоритму

Вдосконалимо отриманий алгоритм таким чином, щоб він відповідав вимогам в пунктах 1, 3 та 5. Почнемо з вимоги, що зазначена в пунткі 1 - розпаралелимо цільову задачу і алгоритм.

3.2.2.1 Розпаралелювання алгоритму

Пункт 1 вимог до алгоритму каже про те, що алгоритм повинен бути розподіленим. Розглянемо початковий алгоритм та проаналізуємо, як можна розпаралелити виконувану алгоритмом задачу і сам алгоритм.

Задача складається з завантаження великої кількості детальних сторінок вакансій. Очевидно, що не має сенсу розбивати саму дію агрегації інформації з однієї конкретної детальної сторінки - на декілька різних кроків, щоб потім їх розпаралелювати. Тобто будемо вважати, що опрацювання однієї конкретної сторінки, після її завантаження - атомарною дією, тобто такою, що не може бути поділена.

Окрім самого опрацювання конкретної сторінки в алгоритмі присутні такі дії як завантаження всіх детальних сторінок та проходження по всім сторінкам головної сторінки. Саме це ми й можемо розпаралелити, оскільки ці задачі не є залежними одна від одної під час виконання задачі, але все ж, задача проходження головної сторінки - передує проходженню детальних сторінок, оскільки саме з головної сторінки ми дізнаємось посилання цих детальних сторінок вакансій.

Але важливо не вдаватись до крайностей і не намагатись розпаралелювати абсолютно кожну дію алгоритму, оскільки в цьому немає сенсу та це не дасть бажаних переваг (час перемикання розрахункових потужностей між задачами - нівелює переваги, не говорячи про значне ускладнення реалізації подібної системи). Основна задача - розподільність системи. Тобто ті атомарні задачі, що будуть мати велику кількість при роботі системи - повинні бути достатньо автономними та мати змогу виконуватись одночасно на декількох серверах - кожна на окремому сервері, незалежно одна від одної.

Зваженим буде розділити задачу та алгоритм наступним чином:

1) один процес - завантажує основну сторінку зі списком вакансій, та пагінує по сторінкам головної сторінки, зберігаючи при цьому посилання на детальні сторінки вакансій, які він “зустрів”.

2) декілька процесів - кожен з процесів опрацьовує одну із збережених першим процесом посилань на детальну сторінку, завантажує її та агрегує відповідну інформацію. Після завершення - процес може перейти до опрацювання наступного посилання.

Якщо сформувати фінальний алгоритм з урахуванням отриманого паралелізму - алгоритм буде задовольняти серед інших вимог - першій вимозі, а саме - алгоритм буде розподіленим.

Але все ще необхідно вирішити питання інших вимог, а саме - вимога 3 про визначення закритих вакансій та вимога 5 - про оптимальність алгоритму. Тому спочатку врахуємо ці вимоги, та лише потім сформуємо фінальний алгоритм.

3.2.2.2 Виявлення закритих вакансій

Яким чином можна визначити, що вакансія закрилась? На різних платформах детальна сторінка закритої або видаленої вакансії може відображатися по різному. Деякі платформи можуть повертати сторінку з інформацією про вакансію з приміткою “Вакансія закрита”, або ж повертати сторінку з повідомленням “Вакансія була видалена”, або ж зовсім повертати сторінку помилки 404 - “Сторінку не знайдено”.

Але є дещо спільне у всіх платформ - закрита та видалена вакансія не відображається в списку вакансій на головній сторінці. І тому, щоб отримана система залишилось гнучкою та мала можливість розширити список джерел - ми будемо використовувати саме цю загальну ознаку закритих або видалених вакансії.

Тому коротко опишемо алгоритм, що виявляє закриття вакансій саме по цій ознаці:

1) Агрегуємо інформацію про всі наявні вакансії.

2) Зберігаємо поточний час (час початку виконання алгоритму) у змінну start_timestamp.

3) Система опрацьовує **всі сторінки** основної сторінки зі списком вакансій та для кожної з вакансій (що є в базі даних) виставляє значення last_seen_at поточним часом.

4) Таким чином, всі ті вакансії в базі даних, що позначені як is_active=True, але мають last_seen_at “старіший” за start_timestamp - це видалені / закриті вакансії, і система виставляє їм значення is_active=False.

У **підпункті 3.2.3.2** цей алгоритм сформованно детальніше.

Таким чином система буде визначати, що раніше агрегована вакансія закрилась. Залишилось лише скомпонувати та оптимізувати всі ці процеси, й описати вдосконалений фінальний алгоритм, що буде задовольняти всім раніше описаним вимогам.

3.2.2.3 Оптимізація процесів

Наразі необхідно проаналізувати та оптимізувати процес агрегації інформації таким чином, щоб система не робила “зайвих” запитів до джерел інформації, а також - мінімізувати затримку між моментом публікації вакансії на ресурсі, до моменту агрегації цієї вакансії системою.

При першому запуску системи ми не маємо змоги зекономити на запитах до ресурсу, оскільки на початку нам необхідно агрегувати всю наявну інформацію. Але при повторній агрегації вже немає потреби повторно агрегувати ті дані, що вже були раніше агреговані системою. З цього випливає декілька оптимізацій:

1) Не агрегувати детальні сторінки вакансій, що вже були агреговані при минулих агрегаціях.

2) Цей пункт випливає з минулого пункту. Враховуючи перший пункт, оскільки вакансії відсортовані за датою публікації, починаючи з якоїсь сторінки - всі вакансії, що будуть знаходитись на сторінці - будуть такими, що були раніше агреговані нами. В такому випадку немає сенсу продовжувати пагінацію по головній сторінці зі списком вакансій.

Але відносно другого пункту - це суперечить алгоритму виявлення закритих вакансій, оскільки для цього нам саме й необхідна пагінація по всім сторінкам. Але все ж, перевіряти наявність нових вакансій - основна задача, що не потребує

великої кількості запитів (після першого циклу агреагції), і нам необхідно мінімізувати час між публікацією вакансії та її агрегацією, тому перевіряти появу нових вакансій необхідно регулярно та доволі часто. Чого не скажеш про виявлення закритих вакансій. Оскільки алгоритм виявлення закритих вакансій потребує пагінації по всім сторінкам, тобто - великої кількості запитів, в залежності від платформи. В такому випадку кількість запитів буде дорівнювати кількості вакансій на платформі поділено на кількість вакансій, що відображається на одній сторінці. Це число буде значити загальну кількість сторінок і, відповідно, кількість необхідних запитів при виявленні закритих вакансій.

Наприклад, при доволі великій платформі, загальна кількість вакансій може досягати 25000 вакансій, при цьому на одній сторінці відображається лише 40 оголошень. Таким чином, платформа матиме 25000/40 = 625 сторінок, що означає що процес виявлення закритих вакансій для цієї платформи буде потребувати 625 запитів до джерела. При чому, на отримання кожної сторінки - потребується час, та можливо необхідно робити затримку між запитами, щоб не перенавантажити джерело, та не отримати бан ір адреси від джерела. Таким чином, якщо на завантаження однієї сторінки і паузу між запитами - виділяти 4 секунди (за умовою послідовних запитів) тоді на пагінацію всіх сторінок знадобиться 625 сторінок * 4 секунди = 2500 секунд = приблизно 41 хвилина. Зауважимо, що це з розрахунком на послідовні запити та на платформу з 25000 оголошень та відображенням 40 оголошень на одній сторінці. Це доволі велика платформа, тому й обрана для ілюстрації не те щоб “найгіршого”, але цілком реального випадку. Й оскільки виявлення закритих вакансій таке “витратне” в плані кількості запитів до ресурсу, й оскільки немає необхідності у частому виконанні цього процесу - з цього всього випливає наступний пункт оптимізації:

3) Розділимо процеси агрегації нових вакансій та виявлення закритих вакансій. Це логічно зробити тому, що метою оптимізації є мінімізація часу затримки агрегації нових вакансій, уникати зайвих запитів на ресурс, та уникати утворення черги на агрегацію детальних сторінок.

Ці три пункта дозволять нам часто виконувати процес агрегації нових вакансій (оскільки він **не** буде потребувати великої кількості запитів), та тим самим мінімізувати затримку між публікацією вакансії та її агрегацією. При цьому, з деякою частотою виконувати процес виявлення видалених вакансій.

3.2.3 Удосконалений алгоритм агрегації вакансій

Таким чином на поточний момент в нас є алгоритм дій для агрегації інформації, є розуміння як цей алгоритм зробити розподіленим, також є алгоритм для виявлення закритих вакансій, та є оптимізації для цих процесів. Отже, врахувавши все це - опишемо фінальний вдосконалений алгоритм, що відповідає поставленим вимогам та являє собою розподілений алгоритм автоматичної агрегації вакансій, що в тому числі агрегує інформацію з детальних сторінок та виявляє видалені вакансії.

3.2.3.1 Алгоритм агрегації вакансій

Опишемо алгоритм агрегації вакансій. Так як алгоритм розподілений, опишемо відповідні дії - як різні процеси.

Агрегація списку посилань на детальні сторінки вакансій, які ще не були агреговані:

1) завантажити “головну” сторінку платформи, що відображає відсортований список вакансій;

2) зберегти у чергу (назвемо її “чергою детальних сторінок на агрегацію”) посилання на детальні сторінки всіх оголошень на поточній сторінці;

3) завантажити наступну сторінку списку вакансій;

4) повторити пункту 2 та 3, доки на черговій сторінці не буде такого набору вакансій, що кожна з них вже була раніше агрегована;

5) періодично повторювати пункти 1 - 4.

Цей процес агрегації списку посилань - можна виконувати окремо для кожної з платформ, що також надає можливість для масштабування.

Тепер сформуємо алгоритм агрегації інформації з детальних сторінок вакансій:

1) очікувати наступне посилання на детальну сторінку вакансії від черги детальних сторінок на агрегацію;

2) отримати посилання на детальну сторінку вакансії з черги детальних сторінок на агрегацію (посилання при цьому видалити з черги);

3) завантажити детальну сторінку вакансії за отриманим посиланням;

4) агрегувати з поточної детальної сторінки всі необхідні дані;

5) повторити пункти 1 - 4.

Ця частина алгоритму може виконуватись одночасно у декількох екземплярах, що дає можливість для масштабування системи.

При цьому, немає необхідності поділу екземплярів на роботу з різними джерелами. Достатньо просто розпізнавати джерело з отриманого посилання, та використовувати відповідний код агрегації/парсингу сторінки.

Sequence-діаграму цього процесу можна знайти у **Додатку В**.

3.2.3.2 Алгоритм виявлення закритих вакансій

Детальніше опишемо алгоритм для виявлення того факту, що раніше агрегована вакансія - була закрита чи видалена з платформи:

1) start_timestamp = поточний час;

2) завантажити “головну” сторінку платформи, що відображає відсортований список вакансій;

3) розпарсити поточну сторінку та отримати список посилань на детальні сторінки вакансій, що відображені на сторінці;

4) для кожної з вакансій - спробувати отримати відповідний запис в базі даних; якщо такий запис наявний - виставити last_seen_at поточним часом;

5) перейти на наступну сторінку списку вакансій;

6) повторити пункту 3 - 5, доки не опрацюємо всі сторінки списку вакансій;

7) порівняти множину “активних” вакансій з множиною раніше агрегованих вакансій. Ті вакансії, що знаходяться в множині раніше агрегованих вакансій, але не знаходяться в множині “активних” вакансій - вважаємо закритими / видаленими.

8) періодично повторювати пункти 1 - 7.

Sequence-діаграму цього процесу можна знайти серед діаграм реалізації системи в **Додатку Ґ**.

Таким чином був сформований алгоритм для розподіленої автоматичної агрегації вакансій, що відповідає заданим вимогам. Наразі необхідно скласти вимоги до імплементації системи, що буде реалізовувати сформований алгоритм. Потім - спроектувати систему враховуючи алгоритми, вимоги до системи, та наявні сучасні технології.

РОЗДІЛ 4 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ РОЗПОДІЛЕНОЇ СИСТЕМИ АВТОМАТИЧНОЇ АГРЕГАЦІЇ ВАКАНСІЙ

4.1 Вимоги до розподіленої системи автоматичної агрегації вакансій

Сформуємо вимоги до кінцевого програмного забезпечення:

1) Система повинна реалізовувати вище сформовані алгоритми для агрегації вакансій, та серед іншого:

- бути розподіленою; [1]
- мати змогу масштабування
- агрегувати детальні сторінки вакансій;
- виявляти видалені вакансії;
- бути автоматичною;

2) Система повинна мати дві платформи в якості джерел інформації. Варто відмітити, що оскільки платформа dou.ua має особливість, що при аналізі була відзначена у відповідному списку пунктом під номером 4, про необхідність виконання JavaScript для отримання списку вакансій - в якості джерел інформації обрано - djinni.co та rabota.ua.

3) Система повинна бути реалізована з використанням засобів віртуалізації, таким чином - бути крос-платформною та мати змогу масштабування. Запуск системи повинен бути скомпонований у одну зручну команду.

4) Цільова база даних повинна зберігати серед іншого наступні дані

- посилання на вакансію;
- детальний опис вакансії;
- дата публікації вакансії;
- дата видалення вакансії;
- компанія роботодавець вакансії;
- платформа, з якої вакансій була агрегована.

5) Система повинна перевіряти наявність нових вакансій на платформах - із заданою параметром частотою.

Варто відмітити, що в рамках цієї роботи ми не ставимо вимог до наявності клієнтського інтерфейса для відображення інформації, а також не реалізуємо механізми завантаження сторінок з використанням технологій, що надають можливість виконання JavaScript (наприклад - Splash). Ці та інші речі - можна реалізовувати в якості подальшого розвитку системи.

4.2 Проектування та реалізація розподіленої системи автоматичної агрегації вакансій

При описі алгоритму агрегації ми не вдавалися у деталі та технічні можливості реалізації сформованих алгоритмів. Цей розділ розрахований саме на це.

Почнемо з підбору сучасних технологій та формування технічної архітектури на основі обраних технологій.

4.2.1 Вибір технологій

Для реалізації цієї системи було обрано об'єктно-орієнтовану інтерпретовану мову програмування загального призначення - Python, версії 3.9. Python є мовою високого рівня зі строгою динамічною типізацією. [6, 8, 9]

Враховуючи всі переваги цієї мови, а також той факт, що ця мова часто використовується для скрапінгу веб-ресурсів і систем агрегації даних [15], та має набір активно підтримуваних спільнотою бібліотек для подібних задач [12] - вибір цієї мови для реалізації нашої мети - цілком логічний.

В якості засобу віртуалізації було обрано технологію **Docker** [2, 5, 16]. Ця сучасна технологія надає чудові засоби віртуалізації, що в свою чергу надасть системі крос-платформності та легкості в масштабуванні. Також ця технологія має

широку підтримку різних сучасних платформ для розгортання коду (Amazon Web Services, Google Cloud Platforms, та інші) [14] та дозволяє використовувати технології аркестрації контейнерів, що в свою чергу надають можливість легкого масштабування системи. Тому, це доцільний вибір для цієї роботи.

В якості веб фреймворку було обрано **Django**. Django - високорівневий відкритий Python-фреймворк (програмний каркас) для розробки веб-систем. [4]

В нашому проекті використовуються такі можливості Django:

- ORM та опис моделей бази даних;
- керування міграціями бази даних;
- можливість подальшого розширення проекту користувацьким web UI.

Тепер постає питання, як реалізовувати розподіленість системи з можливістю масштабування. Саме для таких цілей існує технологія **Celery**. Celery - це асинхронна черга завдань з відкритим кодом або черга завдань, яка базується на розподіленому передаванні повідомлень. Хоча ця технологія підтримує планування, основна увага приділяється операціям у реальному часі. [3]

Celery - розподілена черга завдань. Celery - це проста, гнучка та надійна розподілена система для обробки величезної кількості повідомлень, забезпечуючи при цьому операції інструментами, необхідними для підтримки такої системи. Ця черга завдань, зосереджена на обробці в режимі реального часу, але водночас - підтримує планування завдань. Celery має велику та різноманітну спільноту користувачів та авторів. Celery має відкритий вихідний код і ліцензується за ліцензією BSD License. [3]

Черги завдань використовуються як механізм розподілу роботи між потоками або машинами. В якості вхідних даних для черги завдань виступає одиниця роботи, яка називається завданням. Виділені робочі процеси постійно контролюють черги завдань для виконання нових робіт. Celery спілкується за допомогою повідомлень, зазвичай за допомогою посередника (брокера повідомлень) для посередництва між клієнтами та робітниками. Для ініціювання

завдання клієнт додає повідомлення в чергу, а брокер потім доставляє це повідомлення працівникові. Система Celery може складатися з декількох робітників і посередників, надаючи можливість високої доступності та горизонтальному масштабуванню. Celery по суті надає можливості для реалізації розподілених систем, що і є нашою метою. Тому саме на базі цього фреймворку ми будемо реалізовувати нашу розподілену систему.

В якості брокеру повідомлень для Celery - будемо використовувати **RabbitMQ**. RabbitMQ - брокер повідомлень з відкритим кодом, яке реалізує систему обміну повідомленнями між компонентами програмної системи на основі стандарту AMQP (Advanced Message Queuing Protocol), а також нещодавно з'явилась підтримка плагінів, що надало можливість для підтримки таких протоколів як Streaming Text Oriented Messaging Protocol (STOMP), MQ Telemetry Transport (MQTT), та інших. [10, 17]

RabbitMQ - найпоширеніший брокер повідомлень з відкритим кодом. З десятками тисяч користувачів, RabbitMQ є одним з найпопулярніших брокерів повідомлень з відкритим кодом. RabbitMQ є легким і простим для розгортання як на своїх фізичних серверах так і в хмарах. Він підтримує кілька протоколів обміну повідомленнями. RabbitMQ може бути розгорнутий у розподілених та об'єднаних конфігураціях для задоволення масштабних вимог високої доступності. RabbitMQ працює на багатьох операційних системах та хмарних середовищах та надає широкий спектр інструментів розробника для найбільш популярних мов.

Таким чином, Celery та RabbitMQ - в якості брокеру повідомлень для Celery - являє собою саме тим рішенням, що дозволяє створити розподілену систему. І це рішення цілком підходить під наші потреби при реалізації даного проекту.

В якості реляційної бази даних було обрано **PostgreSQL**.

PostgreSQL - це потужна об'єктно-реляційна система керування базами даних (СКБД) з відкритим кодом, яка активно розробляється вже понад 30 років, що забезпечило їй стійку репутацію надійності та продуктивності. [13]

Для відстежування змін файлів та додаткової зручності при розробці - було використано **Git** - безкоштовну систему розподіленого контролю версій з відкритим кодом, що призначена для швидкого та ефективного управління всіма проектами - від маленьких до величезних. [7]

Отже, усі технології обрані, перейдемо до проектування архітектури системи, на основі обраних технологій.

4.2.2 Побудова архітектури та реалізація розподіленої системи автоматичної агрегації вакансій на основі обраних технологій

Отже, нам необхідно реалізувати розподілену систему за допомогою Celery. Наші алгоритми побудовані на періодичних задачах та на виконанні задач, що генеруються в процесі роботи (при виявленні нової вакансії та посилання на детальну сторінку вакансії - генеруємо задачу на агрегацію даних з цієї детальної сторінки). Планування періодичних задач в Celery досягається за допомогою **Celery Beat**. Celery Beat - це сервіс (процес), що залежно від параметрів - періодично посилає в брокер повідомлень відповідні задачі на виконання. Після цього в роботу вступає **Celery Workers**.

Celery worker - це процес, що підключається до брокеру повідомлень, та очікує на отримання задачі від брокеру, щоб виконати отримане завдання. Таким чином, основними “робітниками” системи виступають саме Celery workers. Суть полягає в тому, що можна створювати багато екземплярів Celery worker, і брокер повідомлень рівномірно розподілить задачі між усіма “працівниками”. Це надає змогу легко масштабувати систему.

Таким чином, за допомогою Celery - досягається розподіленість системи. А за допомогою Celery Beat - досягається автоматичність системи - оскільки система автоматично періодично виконує перевірку наявності нових вакансій, а при виявленні нової вакансії - генерує завдання на агрегацію інформації з детальної сторінки нової вакансії.

Варто відмітити, що при трансформації алгоритмів під обрані технології - “додаванню в чергу посилання на детальну сторінку” - відповідає створення задачі на агрегацію детальної сторінки передаючи посилання на детальну сторінку конкретної вакансії в якості аргументу до задачі.

При побудові системи - одразу використовуємо Docker. Це дозволяє одразу налаштувати мережу між мікро сервісами та надає можливість локально запускати та тестувати систему. Було створено **Dockerfile**, що описує процес побудови **Docker Image**, а саме:

- наслідуємо наш імедж від імеджу **python:3-alpine**;

- оновлюємо та встановлюємо в імеджі всі необхідні оновлення та залежності для ОС;

- копіюємо в імедж всі необхідні файли проекту;

- встановлюємо відповідні python залежності за допомогою **pip** [11] та **requirements.txt**.

Варто зазначити, що код для Celery Beat, Celery Worker, Django web app - знаходяться в одному модулі, тому цілком можна використовувати один і той самий імедж з усіма необхідними файлами проекту коду - для кожного з цих сервісів, і просто для різних сервісів - використовувати відповідні команди запуску, що використовують код у відповідних файлах. Тому, був створений лише один загальний Dockerfile, що копіював всі необхідні файли вихідного коду з розрахунком на те, що імедж буде спільний для всіх сервісів. З отриманого Dockerfile був збудований імедж з назвою “**app**”. Цей імедж використовується сервісами Celery Beat, Celery Worker та Django web app.

Після цього - опишемо **docker-compose.yml** (**Docker Compose**) файл, що дозволяє встановити взаємозалежність між сервісами, об’єднати сервіси в спільну мережу, та запускати систему однією командою (як того потребують вимоги до системи).

Опишемо архітектуру системи, а потім детальніше розглянемо кожен з сервісів та деякі деталі їх реалізації. Архітектура системи складається з наступних сервісів:

- celery_beat;
- celery_worker;
- web_app;
- postgresql;
- rabbitmq.

Розглянемо призначення та реалізацію кожного з сервісів детальніше.

4.2.2.1 Сервіс celery_beat

Сервіс celery_beat - періодично посилає завдання на виявлення нових вакансій, та на оновлення статусу попередньо агрегованих вакансій (перевірку, чи вакансія все ще активна).

Функціонал сервісу складається по суті лише з можливостей Celery, оскільки при старті контейнеру - завантажуються налаштування Celery, що задають регулярність генерації задач на функції:

- **main_vacancy_discovery**;
- **main_update_vacancy_status**;

Розглянемо ці функції детальніше при описі сервісу celery_worker, оскільки саме workers виконують ці функції.

4.2.2.2 Сервіс celery_worker

Сервіс celery_worker являє собою Celery “працівника”, в якому наявна реалізація наступних функцій:

- main_vacancy_discovery;
- main_update_vacancy_status;

Та відповідні функції для кожного з ресурсів (djinni.co та rabota.ua):

- <resource_alias>_vacancy_discovery;
- <resource_alias>_vacancy_detailed_page;
- <resource_alias>_vacancy_status;

де <resource_alias> - унікальний аліас, призначений кожному з ресурсів (для прикладу, один із методів називається - “djinni_vacancy_discovery”).

Діаграми, що описують внутрішню логіку цих функцій та взаємодію між сервісами - можна знайти в **Додатку Ґ**.

Розглянемо детальніше ці функції.

Функція main_vacancy_discovery - витягує з бази даних список всіх активних ресурсів, та для всіх цих ресурсів, посилає (в брокер повідомлень для Celery - RabbitMQ) завдання на виконання відповідної функції <resource_alias>_vacancy_discovery. Після цього RabbitMQ розподілить ці завдання між усіма наявними вільними екземплярами celery_workers. Таким чином, для кожного з ресурсів розподілено виконається відповідна функція vacancy_discovery.

Функція main_update_vacancy_status - аналогічна до main_vacancy_discovery, але замість <resource_alias>_vacancy_discovery, посилає на виконання відповідні функції <resource_alias>_update_vacancy_status.

Функція <resource_alias>_vacancy_discovery - реалізує частину алгоритму, що відповідає за агрегацію списку посилань на детальні сторінки вакансій, що ще не були агреговані (цей алгоритм описаний у першій половині підпункту 3.2.3.1).

На початку, функція завантажує головну сторінку зі списком вакансій та ініціюємо змінну **discovered=0** (ця змінна знадобиться трохи пізніше). Далі для кожної з вакансій, система перевіряє чи є вакансія в базі даних (не наповнення вакансії, а лише запис про неї, без контенту). Якщо запис про вакансію в базі

даних відсутній - система створює цей запис, вказуючи відповідні дані - посилання на вакансію, та **задає last_seen_at поточним часом**. При цьому - система посилає на виконання відповідну <resource_alias>_vacancy_detailed_page функцію для детальної сторінки такої вакансії. Також в такому випадку - система інкрементує значення discovered на 1 (це значення містить в собі кількість “discovered” вакансій - тобто тих нових вакансій, що були вперше виявлені системою).

Якщо ж в базі даних присутній запис про вакансію - ми не відправляємо в брокер повідомлень задачу на детальну сторінку цієї вакансії (оскільки якщо запис наявний, система в минулому вже раніше виявляла цю вакансію та вже створювала відповідну задачу на агрегацію детальної сторінки). Це дозволяє запобігти дублюванню задач на одну й ту саму детальну сторінку вакансії. При цьому функція **все одно оновлює значення last_seen_at відповідної вакансії - поточним часом**.

Після того як всі вакансії на даній сторінці опрацьовані - система використовує змінну discovered, щоб побачити, чи були на даній сторінці нові вакансії. Якщо discovered > 0 - значить на даній сторінці були нові вакансії - система витягує зі сторінки посилання на наступну сторінку - і пагінує на неї, зкидаючи значення discovered в нуль. Якщо ж discovered дорівнює 0, отже на даній сторінці не було нових вакансій, а оскільки список вакансій відсортований за датою за принципом “зверху - новіші вакансії”, отже немає сенсу пагінувати на наступні сторінки, оскільки там “старіші” вакансії, які вже відомі системі та агреговані. І ця логіка відповідно відпрацьовує на кожній наступній сторінці при пагінації. Таким чином система визначає, коли варто зупинити пагінацію.

Блок-схему цього процесу можна переглянути в **Додатку Д**.

Функція <resource_alias>_vacancy_detailed_page - реалізує алгоритм агрегації інформації з детальних сторінок вакансій (описаний у другій половині підпункту 3.2.3.1). Celery worker отримує від RabbitMQ завдання на агрегацію інформації з детальної сторінки для певного ресурсу (посилання на детальну сторінку передається в якості параметру). Сама функція - завантажує сторінку,

парсить її, агрегує всі необхідні дані, та зберігає їх в базу даних. Далі Celery worker очікує наступне завдання (яким може бути як агрегація наступної детальної сторінки, так і будь яке інше завдання, що тут описані). Важливо зазначити, що функція проставляє в базі даних для даної вакансії значення scraped_at та last_seen_at в поточний час.

Функція <resource_alias>_update_vacancy_status - реалізує алгоритм виявлення закритих вакансій, що знаходить у підпункті 3.2.3.2. Функція на початку своєї роботи - запам’ятовує час початку роботи (назвемо це значення **start_timestamp**). Після цього - функція **ітерує по всім сторінкам** (незалежно ні від чого) списку вакансій відповідного ресурсу, та для кожної з вакансій на сторінці, що при цьому наявні в базі даних - **проставляє значення last_seen_at в поточний час**. Одночасно з цим ведеться рахунок кількості таких вакансій в змінній **updated**. А також у змінній **not_scraped_yet** ведеться рахунок вакансій, що є на сайті, але запису про які ще немає в базі даних. Ці змінні не несуть алгоритмічного навантаження, та відображаються лише для статистики. Після того як для всіх вакансій, що наявні на сайті та запис про які є в базі даних - система виставила last_seen_at значення, які відповідно більші за start_timestamp, оскільки вони виставлялись впродовж виконання функції, а start_timestamp був визначений з самого початку виконання функції (найпершим). Задля гарантії того, що start_timestamp менший за всі інші last_seen_at значення, виставлені впродовж роботи функції - ми завчасно віднімемо деяке значення від start_timestamp, щоб start_timestamp гарантовано був менший за будь яке з last_seen_at, що були виставлені вакансіям впродовж роботи функції.

Таким чином, якщо відфільтрувати в базі даних вакансії для даного ресурсу, вибрати лише ті, що наразі мають значення is_active=True, та при цьому last_seen_at яких - менший за start_timestamp - ми отримаємо ті вакансії, що вже не є “активними”, і тому всі ці вакансії необхідно оновити значенням is_active=False. Саме таким чином функція і працює.

Варто зазначити, що час виконання цієї функції залежить від кількості сторінок на ресурсі (оскільки функція пагінує по всім сторінкам), тому функція

main_update_vacancy_status викликається лише раз на добу, і відповідно всі функції <resource_alias>_update_vacancy_status також відпрацьовують лише раз на добу. Це дозволяє уникнути навантаження на веб-ресурс та уникнути отримання системою бану від ресурсу.

Таким чином, це всі методи, наявні у Celery Worker. При розширенні системи новими ресурсами - достатньо додати ресурс у базу даних, та додати реалізацію відповідних функцій <resource_alias>_vacancy_discovery, <resource_alias>_vacancy_detailed_page та <resource_alias>_update_vacancy_status.

4.2.2.3 Сервіс web_app

Цей сервіс являє собою Django web application.

Наразі частина web app має в собі лише архітектурно закладену можливість додання веб-сторінок. В майбутньому функціонал даного web app можна розширити, створивши гарний користувацький інтерфейс, з сортуванням та фільтрацією вакансій. Але наразі веб-інтерфейс по суті відсутній, тому для користування системою необхідно використовувати безпосередньо запити до бази даних.

Цей сервіс також використовується для опису моделей бази даних, для використання ORM, а також - для міграцій бази даних.

4.2.2.4 Сервіс postgresql

Цей сервіс являє собою по суті базу даних PostgreSQL і використовує публічний імедж postgresql. Опишемо моделі бази даних, що реалізовані в проекті:

Таблиця **resource** має такі поля:

- **id** - унікальний автоматично згенерований ідентифікатор ресурсу;
- **alias** - унікальна скорочена назва ресурсу;

- **url** - посилання на сторінку зі списком вакансій;

- **is_active** - бінарне значення того, чи повинна система опрацьовувати даний ресурс;

Таблиця **vacancy** має такі поля:

- **id** - унікальний автоматично згенерований ідентифікатор вакансії;
- **resource_id** - ідентифікатор ресурсу, на якому розміщена вакансія;
- **url** - посилання на детальну сторінку вакансії;
- **internal_id** - ідентифікатор вакансії на ресурсі (можна побачити в посиланні на детальну сторінку вакансії);
- **scraped_at** - дата та час, коли вакансія була агрегована;
- **last_seen_at** - дата та час, коли вакансія востаннє була помічена на сайті;
- **published_at** - дата та час, коли вакансія була опублікована;
- **is_active** - бінерне значення того, чи присутня ця вакансія зараз на сайті;
- **content** - детальний опис вакансії з усією агрегованою інформацією.

Відповідну діаграму моделей можна переглянути у **Додатку Г**.

4.2.2.5 Сервіс rabbitmq

Цей сервіс являє собою екземпляр брокеру повідомлень RabbitMQ, та використовує публічний імедж rabbitmq. Цей сервіс використовується сервісами celery_beat та celery_worker для обміну повідомленнями та таким чином - генерації і розподілення задач.

Також варто зазначити, що сервіси залежні один від одного - наприклад, celery-worker повинен стартувати лише після того як rabbitmq повністю запущенний). Але docker не взмозі це контролювати, оскільки він перевіряє залежність лише в тому плані, що перевіряє чи запустився контейнер із залежним сервісом, але не перевіряє чи встиг залежний сервіс всередині свого контейнера

повністю завершити процес запуску. Тому, для таких цілей був використаний **wait-for.sh bash** скрипт, який передував команді запуску відповідних сервісів, та перевіряв доступність залежних сервісів використовуючи HTTP запит на відповідні url, для перевірки статусу сервісів та впевненості, що залежні сервіси повністю запущені.

Нагадаю, діаграми, що описують внутрішню логіку та взаємодію цих сервісів розподіленої системи автоматичної агрегації вакансій - можна переглянути у **Додатку Ґ**.

4.2.3 Реалізована система, її можливості та результати її роботи

Код реалізованої системи можна знайти у віддаленому репозиторію за посиланням https://github.com/dimabranik/vacancy_aggregator. При розробці системи була закладена можливість запуску системи в різних конфігураціях: для тестування, та для роботи. Конфігурація “для роботи” відрізняється від тестової конфігурації наявністю проксі-сервера (nginx) та запуском web_app через безпечний та високонавантажений WSGI HTTP сервер - Gunicorn, що дозволяє конфігурувати кількість workers, які опрацьовують HTTP запити на сервер. Але оскільки web_app поки що не надає веб-інтерфейс, нам кількість gunicorn workers не є важливою. Перейдемо безпосередньо до способів запуску системи та результатів її роботи.

4.2.3.1 Запуск системи в різних конфігураціях

Отже, перейдемо до інструкцій запуску системи з різними конфігураціями.

4.2.3.1.1 Запуск системи з тестувальною конфігурацією

Запустити систему з тестовою конфігурацією можна у декілька способів.

4.2.3.1.1.1 Запуск системи частково в Docker з тестувальною конфігурацією

Перший спосіб - запуск системи частково на хості (не в Docker), що дозволяє швидше вносити та тестувати зміни в системі.

Варто зазначити, що на хості повинний бути встановлений та запущений Docker. **Всі команди мають виконуватись з кореневої папки проекту.** Та **в кожній новій вкладці терміналу** необхідно першочергово **виконати пункт 2**.

Для цього необхідно:

1) Встановити локально pip залежності.

2) Експортувати env vars з файлу env_files/local.env.

Для цього можна скористатись наступною командою: **source env_files/local.env;export $(cut -d= -f1 env_files/local.env);echo “$POSTGRES_HOST"** - що експортує всі env var з файлу env_files/local.env та в якості підтвердження успіху - виведе на екран значення env var **POSTGRES_HOST**, що буде дорівнювати **localhost**. Цей крок необхідно буде робити для кожної з вкладок терміналу, які будуть використані під час запуску системи.

3) Запустити **postgres** сервіс в Docker та провести міграцію бази даних.
Це можна зробити наступними командами:

- docker run --env-file env_files/local.env -v "$PWD/docker/postgres_storage/:/var/lib/postgresql/data" -p "5432:5432" postgres

- python manage.py migrate

3.1) (Лише при першому запуску) Необхідно додати в базу даних відповідні записи про ресурси djinni.co та rabota.ua за допомогою наступних SQL запитів до бази:

INSERT INTO public.collector_app_resource (alias, url, is_active) VALUES ('rabota_ua', 'https://rabota.ua/zapros/all/украина?parentId=1', true);

INSERT INTO public.collector_app_resource (alias, url, is_active) VALUES ('djinni_co', 'https://djinni.co/jobs/', true);

Host, port, database, username, та password - можна переглянути у env_files/local.env файлі)

4) Запустити **rabbitmq** сервіс в Docker:

- docker run --env-file env_files/local.env -v "$PWD/docker/rabbitmq_storage/rabbitmq:/var/lib/rabbitmq" -p "5672:5672" rabbitmq:3

5) Запустити **celery_beat** проце**с:**

- celery -A vaca_project beat -l info --scheduler django_celery_beat.schedulers:DatabaseScheduler

6) Запустити необхідну кількість **celery_workers**:

- celery -A vaca_project worker -l info --concurrency=1 -n celery_worker_N

Варто зауважити, що для кожного з воркерів можна вказати параметр — **concurrency**, що означає кількість потоків у воркері, які будуть працювати над задачами (є один “головний” потік, який слухає повідомлення від брокеру повідомлень, а всі інші потоки у кількості —**concurrency** - працюють над задачами, що отримав “головний” потік). Тобто, **кожен з воркерів може працювати в декілька потоків.**

Для запуску декількох воркерів - можна повторно використати цю ж команду, але варто змінити останній аргумент (**-n**) на унікальний - наприклад, ітерувати N в кінці **celery_worker_N** від 1 і далі, щоб воркери отримували відповідно імена celery_worker_1, celery_worker_2, і тд.

Важливо те, що **celery_workers самі відслідковують наявність інших воркерів, та автоматично синхронізуються**.

7) Запустити web_app:

- python manage.py migrate && python manage.py runserver 0.0.0.0:8000

4.2.3.1.1.2 Запуск системи повністю в Docker з тестувальною конфігурацією

Інший спосіб запуску системи - запуск всієї системи в Docker. Зробити це однією командою, як того потребують вимоги до реалізації системи. Відмінність від першого підходу - час запуску системи збільшується, що заважає під час

активної розробки системи. Але після того як система готова - такий спосіб запуску для локального тестування системи - є кращим та зручнішим. Команда запуску: **docker-compose up**

Як і в минулому разі, не хості повинен бути встановлений та запущений Docker. І при першому запуску необхідно додати відповідні ресурси до бази даних, як це робиться у пункті 3.1 підпункту 4.2.3.1.1.1.

4.2.3.1.2 Запуск системи з PRODUCTION конфігурацією

Запуск системи з PRODUCTION конфігурацією, як вже зазначалось вище, відрізняється тим, що додається декілька web servers задля безпеки web_app сервісу, а також використанням іншого файлу налаштувань системи. Досягається це за рахунок перевизначення деяких значень в docker-compose.yml файлі. Команда запуску системи в цьому випадку виглядає наступним чином:

docker-compose -f docker-compose.yml -f docker-compose.prod.yml up

4.2.3.2 Результати роботи системи, можливості, та обмеження

Після запуску системи, celery_beat витягне налаштування з файлу vaca_project/settings/settings.py (vaca_project/settings/production.py у випадку запуску системи з PRODUCTION конфігурацією) та почне відправляти до брокера повідомлень задачі main_vacancy_discovery та main_vacancy_status_update. celery_worker отримає ці задачі і почне їх виконувати, генеруючи нові задачі, і виконуючи їх. Згодом, в базу даних почнуть надходити агреговані вакансії, що можна переглядати, використовуючи будь-який database IDE з підтримкою PostgreSQL.

Приклад результату роботи системи (агреговані вакансії) можна переглянути у **Додатку Е**.

Важливо відмітити, що так як система “Докеризована” (тобто загорнута в docker image та може бути запущена в docker) - це дозволяє використовувати

Docker Container Orchestration системи - такі як Kubernetes, які надають можливість розгортати застосунок одразу на багатьох кластерах.

Варто також відмітити, що навіть при локальному запуску командою docker-compose up - використовується можливість docker-compose та запускаються одразу два celery_worker (це можна конфігурувати в docker-compose replicas).

До того ж, самі воркери є багатопоточними, що надає додаткової гнучкості та потужності системі.

Говорячи про недоліки системи - не можна сказати що є якийсь очевидний недолік саме архітектури системи. Однак, поточна реалізація системи не бездоганна, і є багато речей, що значно розширили б можливості системи.

Серед недоліків системи варто зазначити наступне:

- відсутність проксі - це створює ризик бану ip адреси системи, а також розкриваючи ip адресу системи створює можливість DDOS-атаки на систему;

- виконання <resource_alias>_vacancy_status при поточній реалізації займає відносно багато часу в залежності від розміру ресурсу; ця функція також може бути розділена на паралельні задачі;

- в системі не використовуються такі інструменти як Splash, що надали б можливість завантажувати сторінки ресурсів, що потребують JavaScript;

- не вирізається html tags та JS код з контенту вакансій, що система зберігає в базу даних; якщо в майбутньому відображати цей контент на веб-сторінці через web_app, це може створювати певну небезпеку для користувачів.

ВИСНОВКИ

В результаті виконання роботи були досліджені платформи з розміщення вакансій та сформована картина типової платформи. Також були сформовані вимоги до алгоритму та створений розподілений алгоритм автоматичної агрегації інформації з цих ресурсів, а також алгоритм оновлення статусу вакансій (виявлення закритих вакансій). Після цього були сформовані вимоги до системи, що реалізовувала б сформовані алгоритми. Після - була побудована архітектура системи на основі сучасних технологій, що б задовільняла вимоги до реалізації системи. На основі цієї архітектури була створена розподілена система автоматичної агрегації інформації. Отримана система є розподіленою та піддається масштабуванню. Також система є досить гнучкою та дозволяє розширення списку ресурсів без змін існуючої архітектури системи. Результуюча система вирішує задачу розподіленої автоматичної агрегації вакансій з різних веб-ресурсів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Addison Wesley. Distributed Systems Concepts and Design. 2001.

2. Barbier, Julien. It's Here: Docker 1.0. Docker. Docker, Inc. 2014.

3. Celery: Distributed Task Queue. 2021. URL: https://docs.celeryproject.org/en/stable/

4. Django. 2021. URL: https://www.djangoproject.com

5. Docker overview. 2021. URL: https://docs.docker.com/get-started/overview/

6. General Python FAQ - Python 3.9.2 documentation. 2021. URL: docs.python.org.

7. Git. 2021. URL: https://git-scm.com

8. Guido van Rossum. Python Reference Manual, release 2.4.4. 2006.

9. Guttag, John V. Introduction to Computation and Programming Using Python: With Application to Understanding Data. MIT Press. 2016.

10. Launch of RabbitMQ Open Source Enterprise Messaging. 2007. URL: https://www.rabbitmq.com/resources/RabbitMQ_PressRelease_080207.pdf

11. Pip. 2021. URL: https://pip.pypa.io/en/stable/

12. Popular Python Libraries to Perform Web Scraping. 2020. URL: https://www.analyticsvidhya.com/blog/2020/04/5-popular-python-libraries-web-scraping/

13. PostgreSQL: The World's Most Advanced Open Source Relational Database. 2021. URL: http://www.postgresql.org

14. The Shortlist of Docker Hosting. 2016. URL: https://www.cloudbees.com/blog/the-shortlist-of-docker-hosting/

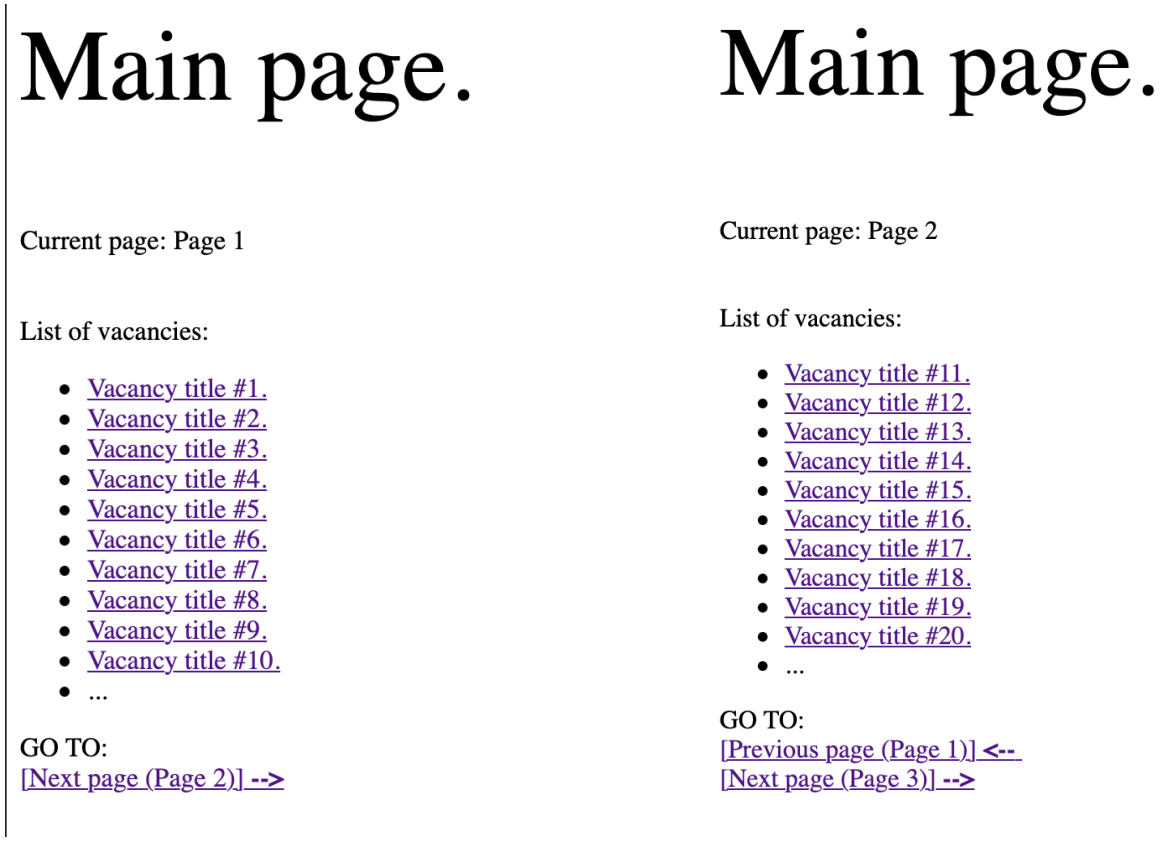
15. What are The Best Programming Languages for Web Scraping? 2020. URL: https://www.promptcloud.com/blog/best-programming-language-for-web-scraping/

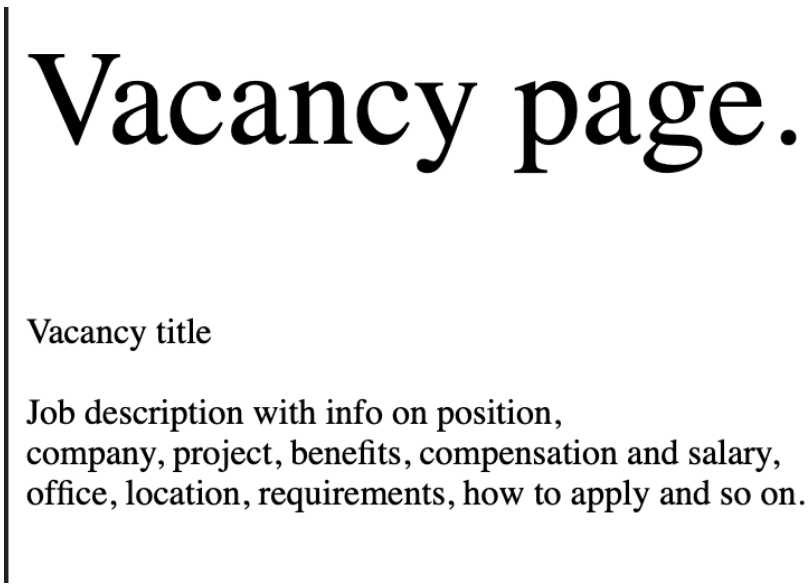
16. What is a Container?. Docker, Inc. 2019. URL: docker.com

17. Which protocols does RabbitMQ support? 2021. URL: https://www.rabbitmq.com/protocols.html

ДОДАТОК А

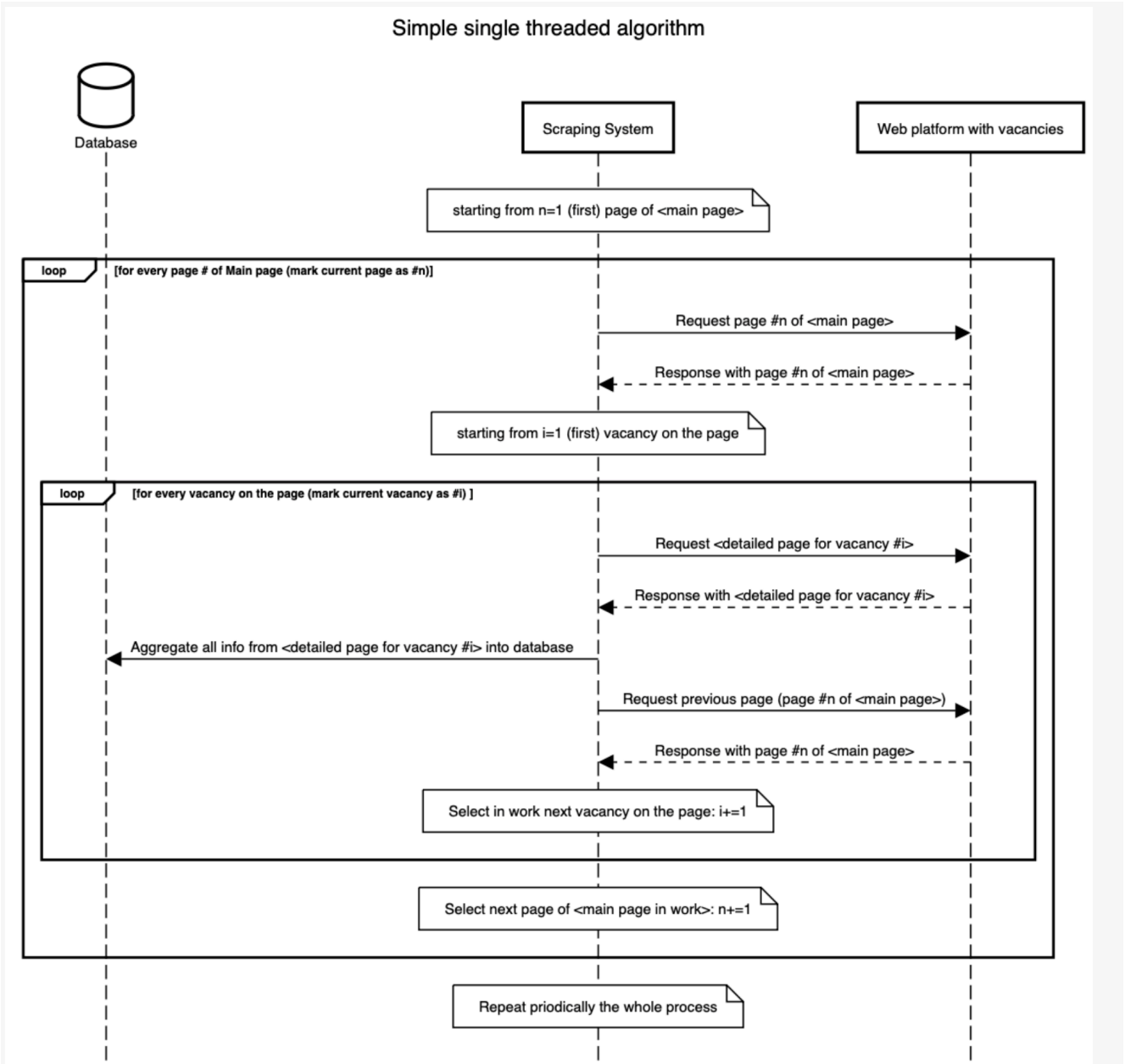
Схематична структура типової платформи з розміщення вакансій





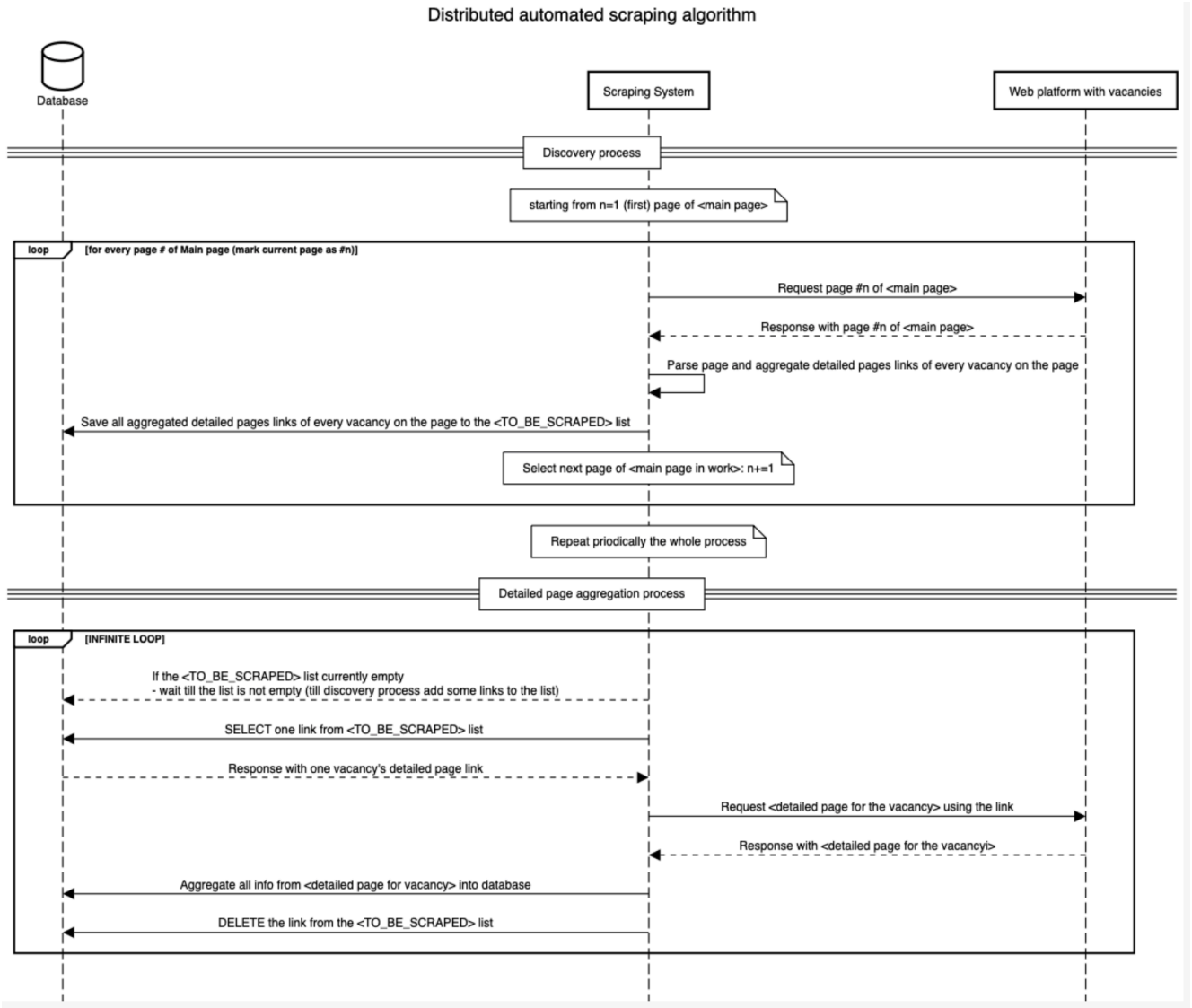
ДОДАТОК Б

Sequence-діаграма початкового алгоритму агрегації вакансій



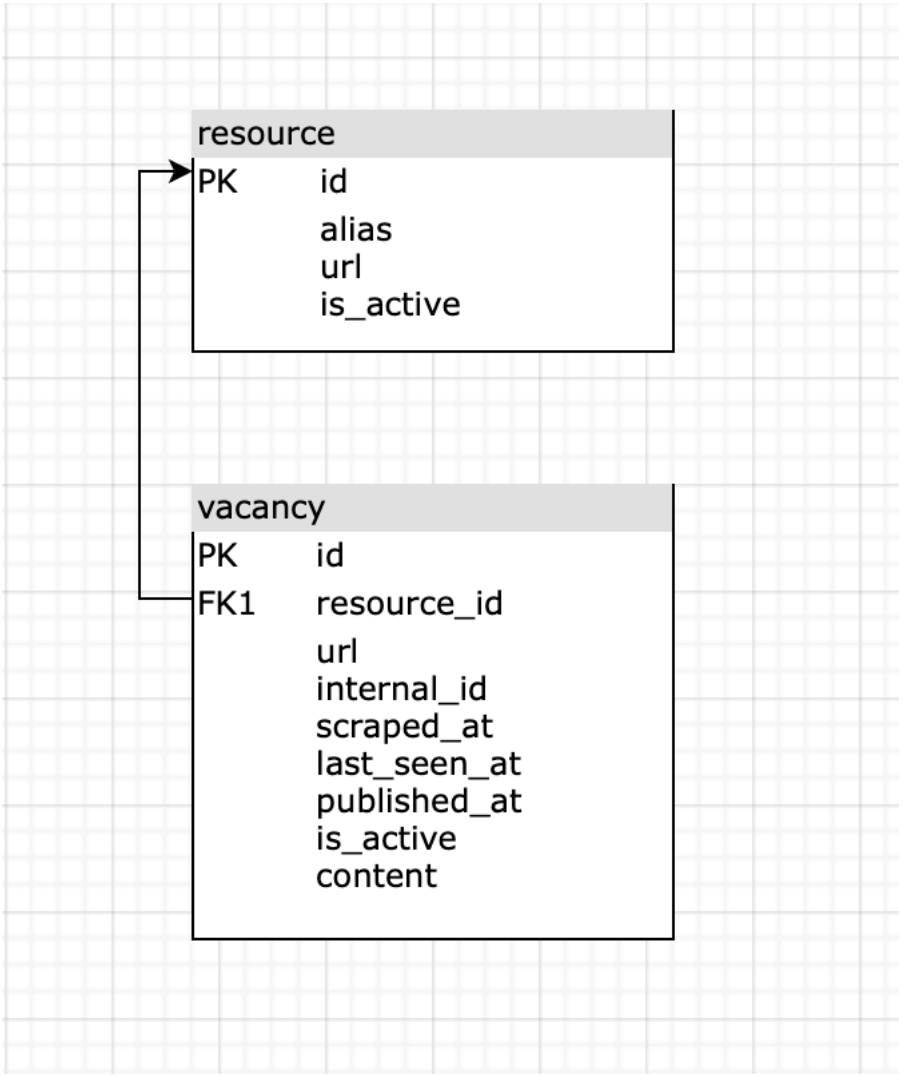
ДОДАТОК В

Sequence-діаграми розподіленого алгоритму автоматичної агрегації вакансій



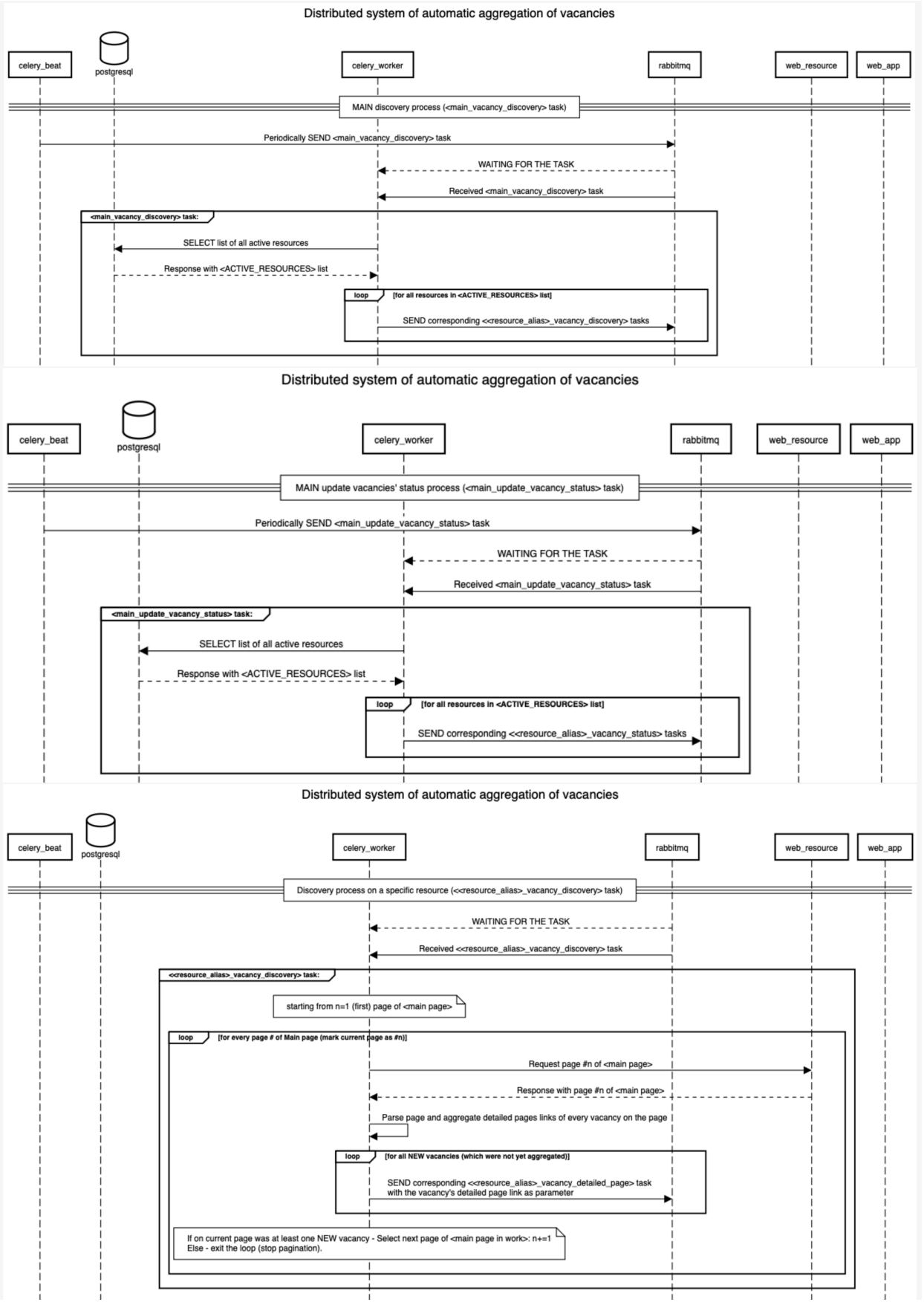
ДОДАТОК Г

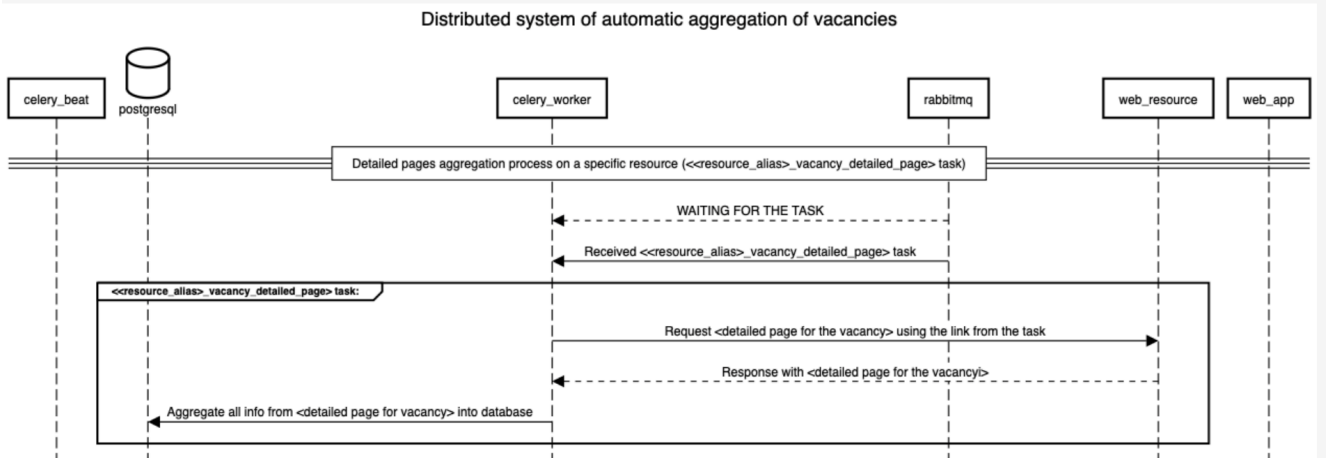
UML-діаграма моделей бази даних розподіленої системи автоматичної агрегації вакансій

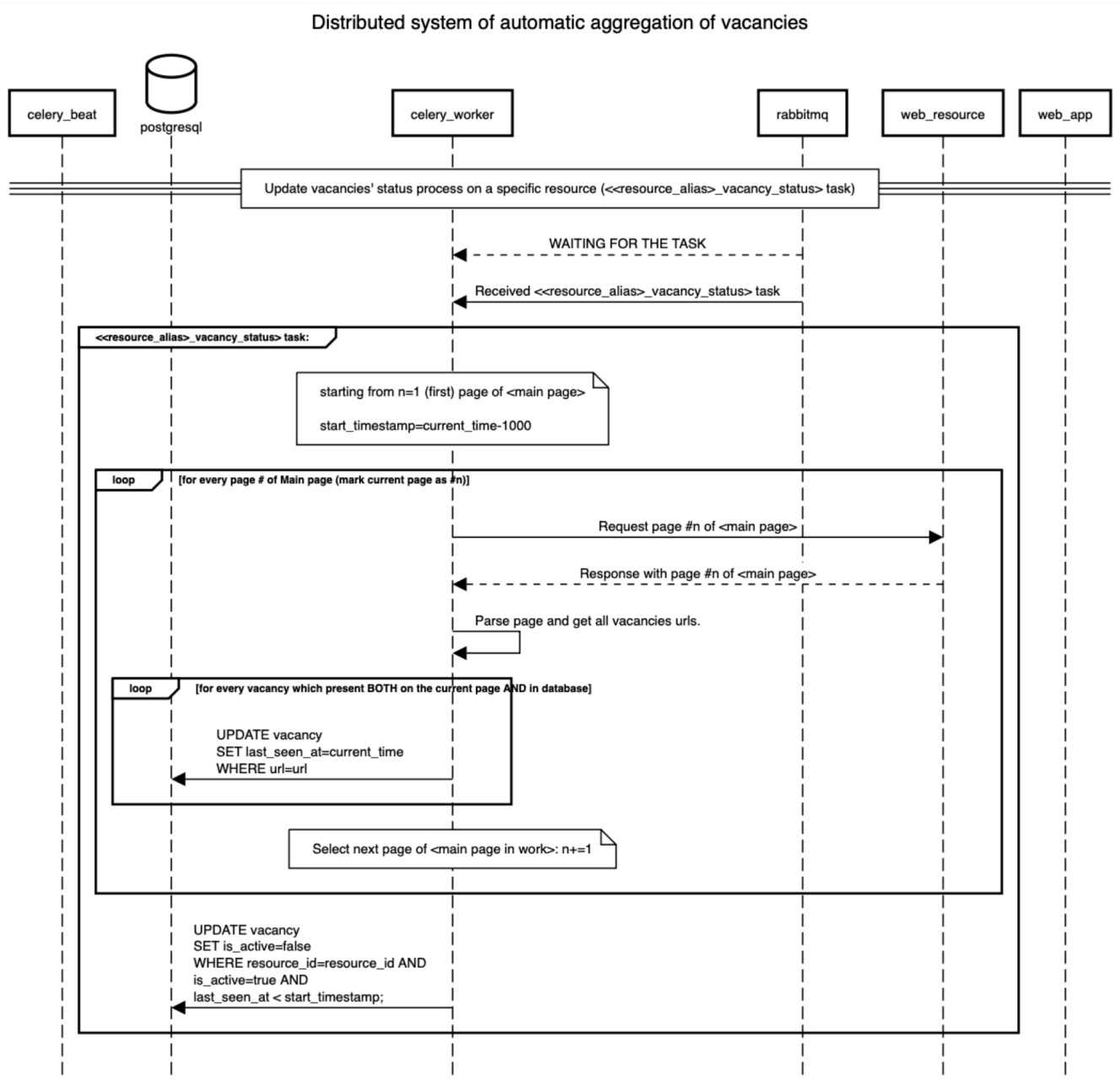


ДОДАТОК Ґ

Sequence-діаграма розподіленої системи автоматичної агрегації вакансій

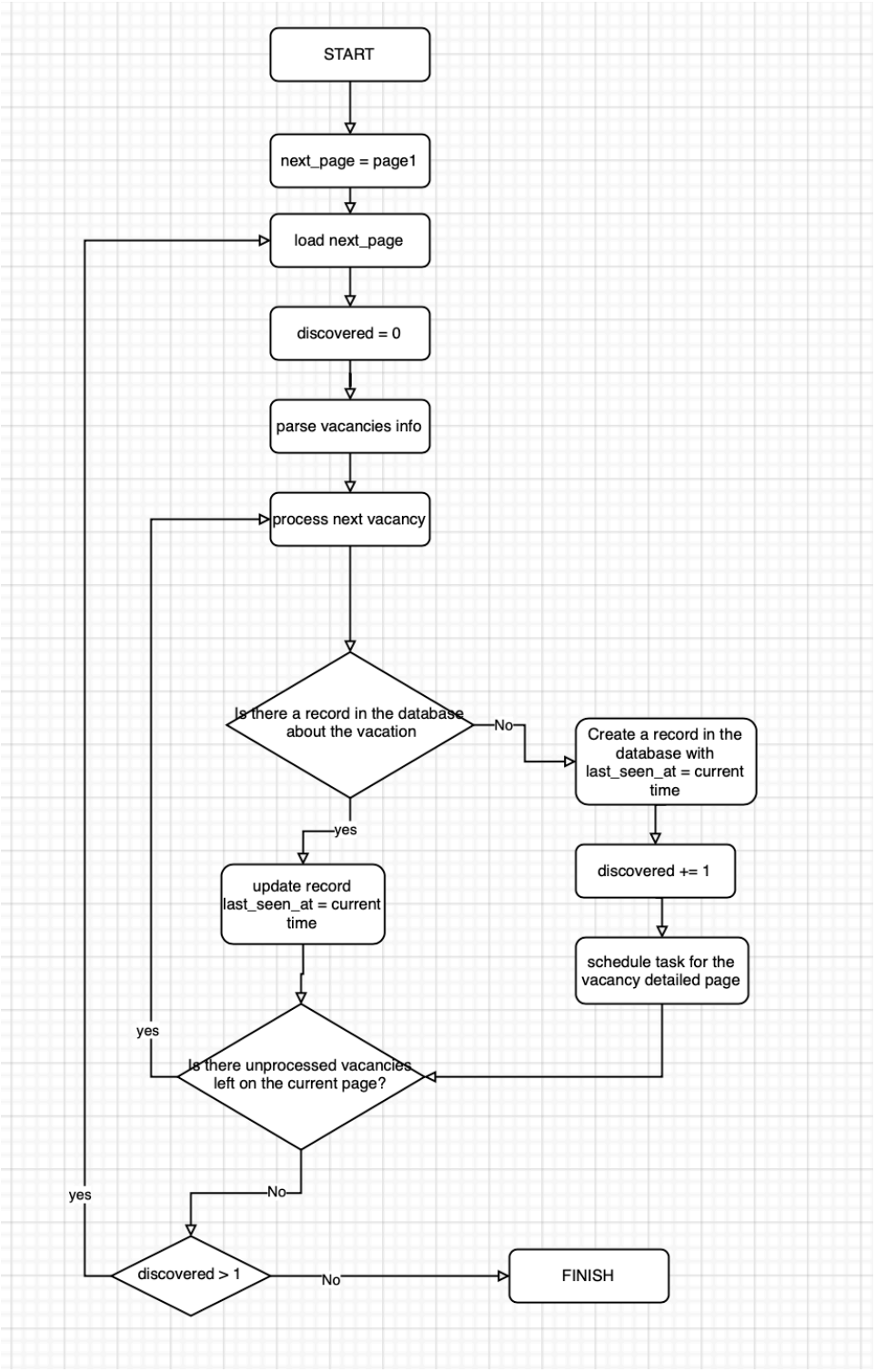



Distributed system of automatic aggregation of vacancies
celery_beat
postgresql
celery_worker
rabbitmq
web_resource
web_app
Detailed pages aggregation process on a specific resource (<<resource_alias>_vacancy_detailed_page> task)
WAITING FOR THE TASK
Received <<resource_alias>_vacancy_detailed_page> task
<<resource_alias>_vacancy_detailed_page> task:
Request <detailed page for the vacancy> using the link from the task
Response with <detailed page for the vacancy!>
Aggregate all info from <detailed page for vacancy> into database


Distributed system of automatic aggregation of vacancies
celery_beat
postgresql
celery_worker
rabbitmq
web_resource
web_app
Update vacancies' status process on a specific resource (<<resource_alias>_vacancy_status> task)
WAITING FOR THE TASK
Received <<resource_alias>_vacancy_status> task
<<resource_alias>_vacancy_status> task:
starting from n=1 (first) page of <main page>
start_timestamp=current_time-1000
loop
[for every page # of Main page (mark current page as #n)]
Request page #n of <main page>
Response with page #n of <main page>
Parse page and get all vacancies urls.
loop
[for every vacancy which present BOTH on the current page AND in database]
UPDATE vacancy
SET last_seen_at=current_time
WHERE url=url
Select next page of <main page in work>: n+=1
UPDATE vacancy
SET is_active=false
WHERE resource_id=resource_id AND
is_active=true AND
last_seen_at < start_timestamp;

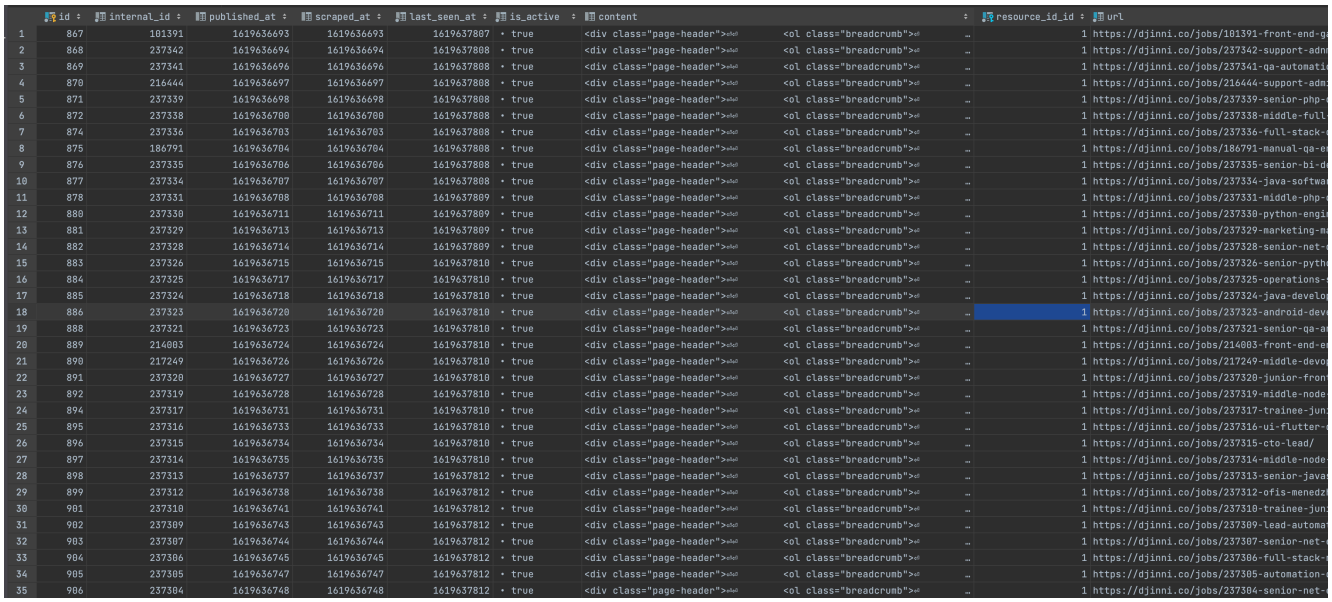
ДОДАТОК Д

Блок-схема роботи функцій <resource_alias>_vacancy_discovery



ДОДАТОК Е

Агреговані вакансії



	id	internal_id	published_at	scraped_at	last_seen_at	is_active	content	resource_id	url
1	867	101391	1619636693	1619636693	1619637807	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/101391-front-end-g
2	868	237342	1619636694	1619636694	1619637808	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237342-support-adn
3	869	237341	1619636696	1619636696	1619637808	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237341-qa-automati
4	870	216444	1619636697	1619636697	1619637808	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/216444-support-adm
5	871	237339	1619636698	1619636698	1619637808	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237339-senior-php-
6	872	237338	1619636700	1619636700	1619637808	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237338-middle-full
7	874	237336	1619636703	1619636703	1619637808	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237336-full-stack-
8	875	186791	1619636704	1619636704	1619637808	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/186791-manual-qa-e
9	876	237335	1619636706	1619636706	1619637808	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237335-senior-bi-d
10	877	237334	1619636707	1619636707	1619637808	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237334-java-softwa
11	878	237331	1619636708	1619636708	1619637809	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237331-middle-php-
12	880	237330	1619636711	1619636711	1619637809	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237330-python-engi
13	881	237329	1619636713	1619636713	1619637809	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237329-marketing-m
14	882	237328	1619636714	1619636714	1619637809	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237328-senior-net-
15	883	237326	1619636715	1619636715	1619637810	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237326-senior-pyth
16	884	237325	1619636717	1619636717	1619637810	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237325-operations-
17	885	237324	1619636718	1619636718	1619637810	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237324-java-develo
18	886	237323	1619636720	1619636720	1619637810	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237323-android-dev
19	888	237321	1619636723	1619636723	1619637810	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237321-senior-qa-a
20	889	214003	1619636724	1619636724	1619637810	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/214003-front-end-e
21	890	217249	1619636726	1619636726	1619637810	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/217249-middle-devo
22	891	237320	1619636727	1619636727	1619637810	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237320-junior-fron
23	892	237319	1619636728	1619636728	1619637810	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237319-middle-node
24	894	237317	1619636731	1619636731	1619637810	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237317-trainee-jun
25	895	237316	1619636733	1619636733	1619637810	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237316-ui-flutter-
26	896	237315	1619636734	1619636734	1619637810	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237315-cto-lead/
27	897	237314	1619636735	1619636735	1619637810	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237314-middle-node
28	898	237313	1619636737	1619636737	1619637812	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237313-senior-java
29	899	237312	1619636738	1619636738	1619637812	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237312-ofis-menedz
30	901	237310	1619636741	1619636741	1619637812	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237310-trainee-jun
31	902	237309	1619636743	1619636743	1619637812	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237309-lead-automa
32	903	237307	1619636744	1619636744	1619637812	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237307-senior-net-
33	904	237306	1619636745	1619636745	1619637812	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237306-full-stack-
34	905	237305	1619636747	1619636747	1619637812	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237305-automation-
35	906	237304	1619636748	1619636748	1619637812	true	<ol class="breadcrumb">	1	https://djinni.co/jobs/237304-senior-net-