

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Сайт працевлаштування у сфері ІТ-технологій»

Виконав студент групи ПЗ-19-2
ТУРОВСЬКИЙ Ярослав Сергійович

Керівник роботи:
ЛОБАНЧИКОВА Надія Миколаївна

Рецензент:
МОРОЗОВ Андрій Васильович

Житомир – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

В.о. зав. кафедри інженерії
програмного забезпечення
Андрій МОРОЗОВ
«14» лютого 2023 р.

ЗАВДАННЯ
на випускню кваліфікаційну роботу

Студента ТУРОВСЬКОГО Ярослава Сергійовича

Тема роботи: «Сайт працевлаштування у сфері ІТ-технологій»,
затверджена Наказом університету від «14» лютого 2023 р., №70с

Вихідні дані для роботи: Методи та засоби розробки веб-сайту за визначеними предметними областями. Моделі, веб-орієнтовані технології, структури, засоби та методи розробки та взаємодії веб-сайтів та модулів управління сайту.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Лобанчикова Н.М.	16.02.2023р.	16.02.2023р.
2	Лобанчикова Н.М.	16.02.2023р.	16.02.2023р.
3	Лобанчикова Н.М.	16.02.2023р.	16.02.2023р.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі	01.03.2023 - 01.04.2023	виконано
2	Пошук, огляд та аналіз додатків для реалізації	02.04.2023 - 05.04.2023	виконано
3	Формулювання технічного завдання	06.04.2023 - 10.04.2023	виконано
4	Опрацювання літературних джерел	11.04.2023 - 15.04.2023	виконано
5	Проектування структури	16.04.2023 - 20.04.2023	виконано
6	Написання програмного коду	21.04.2023 - 10.05.2023	виконано
7	Відлагодження	11.05.2023 - 17.05.2023	виконано
8	Написання пояснювальної записки	18.05.2023 - 25.05.2023	виконано

РЕФЕРАТ

Випускна робота бакалавра на тему «Сайт працевлаштування в сфері ІТ-технологій» складається з вступу, трьох розділів, висновків, списку використаної літератури та додатку.

Текстова частина викладена на 62 сторінках друкованого тексту. Пояснювальна записка має 19 сторінок додатків. Список використаних джерел містить 18 найменувань і займає 2 сторінки. В роботі наведено 48 рисунків та 17 таблиць. Загальний обсяг роботи – 81 сторінка.

Метою роботи є розробка та впровадження функціонального та ефективного веб-сайту, спрямованого на допомогу людям у пошуку роботи в ІТ сфері. Цей продукт має на меті забезпечити користувачів зручним функціоналом як для пошуку працівників, роботодавцями, так і для пошуку вакансій, людьми що знаходяться в стані пошуку роботи.

В результаті розробки було створено програмний продукт, який повністю готовий до експлуатації та може служити надійною основою для подальшого розширення. Завдяки використанню Docker, проект з легкістю можна розвернути на будь якій UNIX-подібній операційній системі. Цей проект може бути використаний як каркас для розробки більш масштабних проектів, які відрізнятимуться надійним рівнем безпеки, високою продуктивністю, зручністю використання та можливістю масштабування.

КЛЮЧОВІ СЛОВА: ВЕБДОДАТОК, API, DOCKER, REACT, SYMFONY.

					ІПЗ.КР.Б – 121 – 23 – ПЗ				
Змн.	Арк.	№ докум.	Підпис	Дата	Сайт працевлаштування у сфері ІТ-технологій	Літ.	Арк.	Аркуші	
Розроб.		Я. Туровський							
Керівник		Н.Лобанчикова					4	81	
Реценз		А. Морозов				Житомирська політехніка, група ІПЗ-19-2			
Н. контр.		О. Тичина							
Зав. каф.		А. Морозов			Пояснювальна записка				

ABSTRACT

The bachelor's thesis on the topic "Employment Website in the IT Technology Field" consists of an introduction, three chapters, conclusions, a list of references, and an appendix.

The textual part is presented on 62 pages of the printed text. The explanatory note comprises 19 pages of appendices. The list of references includes 18 sources and spans 2 pages. The thesis includes 48 figures and 17 tables. The total length of the thesis is 81 pages.

The aim of this work is to develop and implement a functional and efficient website aimed at assisting people in their job search in the IT field. This product aims to provide users with convenient functionality for both job seekers and employers, facilitating the search for employees and job opportunities.

As a result of the development, a software product has been created that is fully ready for operation and can serve as a reliable foundation for future expansion. By utilizing Docker, the project can be easily deployed on any UNIX-like operating system. This project can serve as a framework for the development of larger-scale projects characterized by a high level of security, performance, usability, and scalability.

KEY WORDS: API, DOCKER, REACT, SYMFONY, WEB APPLICATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ САЙТУ ПРАЦЕВЛАШТУВАННЯ В СФЕРІ ІТ-ТЕХНОЛОГІЙ	10
1.1. Постановка задачі.....	10
1.2. Аналіз аналогів програмного продукту	11
1.3. Вибір архітектури сайту з працевлаштування в сфері ІТ-технологій	14
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	16
Висновки до першого розділу	19
РОЗДІЛ 2. ПРОЕКТУВАННЯ САЙТУ З ПРАЦЕВЛАШТУВАННЯ У СФЕРІ ІТ-ТЕХНОЛОГІЙ	20
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи	20
2.2. Розробка бази даних сайту з працевлаштування у сфері ІТ-технологій ...	24
2.3. Проектування та реалізація алгоритмів роботи системи	32
2.4. Реалізація сайту працевлаштування у сфері ІТ-технологій.....	34
Висновки до другого розділу	37
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З САЙТОМ ПРАЦЕВЛАШТУВАННЯ У СФЕРІ ІТ-ТЕХНОЛОГІЙ	38
3.1. Порядок встановлення та налаштування параметрів системи.....	38
3.2. Структура інтерфейсу сайту з працевлаштування у сфері ІТ-технологій	38
3.3. Тестування роботи сайту	52
Висновки до третього розділу	57
ВИСНОВКИ	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	60
ДОДАТКИ	62

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІТ – Інформаційні технології.

API – Application Programming Interface.

UI – User interface.

CPU – Central Processing Unit.

БД – База даних.

СУБД – Система управління базами даних.

ORM – Object-Relational Mapping.

ВСТУП

Актуальність теми. ІТ сфера швидко розвивається і вимагає все більше кваліфікованих фахівців. Багато компаній постійно шукають нові таланти в галузі програмування, розробки веб-додатків, кібербезпеки, аналізу даних та інших спеціалізацій. Сайт пошуку роботи в ІТ сфері може стати цінним ресурсом для співробітників та роботодавців, які шукають взаємовигідних партнерів.

На даний момент, на ринку існує багато різних сайтів з пошуку роботи, але більшість з них є універсальними, що ускладнює пошук роботи для фахівців вузьких спеціальностей. Сфера інформаційних технологій може бути дуже специфічною і вимагати конкретних знань та якостей. Фахівцям даної сфери потрібно вказувати свої навички в технічних напрямках, досвід та вподобання, а роботодавцям шукати відповідних кандидатів для конкретних ролей. Тому розробка спеціалізованого сайту пошуку роботи в ІТ сфері може задовольнити ці потреби і забезпечити зручну та ефективну платформу для співпраці між роботодавцями та фахівцями.

На сайті працівники зможуть зробити своє детальне резюме, та розпочати пошук потрібних їм вакансій, або ж чекати на момент коли їх резюме зацікавляться компанії та напишуть першими. В свою чергу представникам компаній буде надано широкий функціонал створення вакансій та пошуку фахівців.

Метою випускної роботи є проектування та розробка сайту з пошуку роботи в ІТ сфері засобами Symfony, React, Docker та MySQL, для полегшення пошуку вакансій та пошуку фахівців в сфері ІТ-технологій.

Поставлена мета обумовлює необхідність виконання наступних завдань:

- провести аналіз напрямків використання інформаційних технологій для розробки сайту працевлаштування в сфері іт-технологій;
- провести аналіз аналогів програмних продуктів, що представлені на ринку;

- обґрунтувати вибір архітектури сайту, інструментальні засоби;
- визначити вимоги до апаратного забезпечення;
- спроектувати сайт з працевлаштування у сфері ІТ-технологій;
- провести програмну реалізацію запропонованого проєкту;
- провести тестування програмного продукту.

Об’єктом дослідження є інформаційні технології створення веб-сайтів.

Предметом дослідження є моделі, методи, алгоритми та засоби створення веб-сайтів прикладного спрямування.

В роботі використано методи об’єктно-орієнтованого проектування та об’єктно-орієнтованого програмування веб-додатків.

Запропонована інформаційна система спроектована, протестована та готова до експлуатації. Область застосування: використання вебплатформи для працевлаштування громадян України в сфері ІТ.

Випускна робота складається з вступу, 3-х розділів, висновків, списку використаних джерел із 18 найменувань (2 стор.), 2 додатків (19 стор.), 17 таблиць та 48 ілюстрацій по тексту. Повний обсяг роботи становить 81 сторінка, з них 62 сторінок основного тексту.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ САЙТУ ПРАЦЕВЛАШТУВАННЯ В СФЕРІ ІТ-ТЕХНОЛОГІЙ

1.1. Постановка задачі

Сайт що розробляється, призначений для пошуку вакансій або спеціалістів у сфері інформаційних технологій. Даний продукт покращує досвід користувачів і приносить ряд переваг, зокрема:

- надає зручну та централізовану платформу, де роботодавці можуть розмішувати вакансії, а кандидати можуть шукати і відгукатися на них. Це дозволяє зекономити час і зусилля, оскільки користувачам не потрібно переглядати безліч окремих джерел або оголошень;
- надає розширені функції пошуку та фільтрації, які допомагають представникам компаній знайти фахівців, що відповідають їхнім конкретним потребам і вимогам. Це може включати фільтрацію за місцем роботи, заробітною платою, навичками, рівнем досвіду та іншими параметрами;
- надає можливості спілкування та взаємодії між кандидатами та роботодавцями, наприклад, через онлайн-чат. Це дозволяє швидко обмінюватися інформацією, задавати питання та проводити співбесіди безпосередньо на платформі.

Основними кроками реалізації такого проекту є:

1. Збір вимог та аналіз потреб користувачів. Це включає визначення основних функцій та можливостей сайту, які бажають мати користувачі, а також встановлення технічних вимог до розробки.
2. Проектування інтерфейсу користувача (UI) та розробка дизайну. На цьому етапі розробляється вигляд та функціональність сайту, включаючи розташування елементів, структуру сторінок, пошукові функції та фільтри.

3. Розробка функціоналу. Після проектування UI настає етап розробки функціональності. Це включає:
 - створення бази даних для зберігання даних;
 - реєстрація фахівців та роботодавців;
 - авторизація;
 - функціонал резюме;
 - функціонал компаній, включаючи покупку тарифних планів;
 - функціонал вакансій;
 - функціонал переписки між роботодавцями та фахівцями.
4. Наповнення сайту контентом для демонстрації та тестування
5. Тестування сайту для перевірки його функціональності, відповідності вимогам та виявлення помилок або дефектів. Тестування включає перевірку різних сценаріїв взаємодії користувачів з сайтом, перевірку правильності роботи пошуку та фільтрів, а також перевірку безпеки та захисту даних

Результатом розробки є функціонуючий вебсайт, який надає користувачам зручний доступ до вакансій в ІТ сфері, можливість відфільтрувати вакансії за різними критеріями, розміщувати резюме, спілкуватися з роботодавцями та знаходити нові робочі місця

1.2. Аналіз аналогів програмного продукту

В Україні існує кілька популярних вебплатформ, спеціалізованих на пошуку роботи в ІТ сфері. Ці сайти надають широкий вибір вакансій та створюють зручне середовище для спілкування між кандидатами та роботодавцями. Наявність таких аналогів свідчить про великий попит на робочі місця в ІТ секторі та необхідність у зручних інструментах для пошуку і знаходження вакансій.

DOU [1] є одним з найпопулярніших та найвпливовіших веб-ресурсів для пошуку роботи в ІТ галузі в Україні. Він відомий своєю великою спільнотою

професіоналів, яка активно обговорює технології, тренди, новини та практичні аспекти роботи в ІТ.

DOU пропонує розділ вакансій, де компанії можуть публікувати оголошення про роботу, а кандидати можуть шукати вакансії за різними критеріями, такими як місце розташування, тип роботи та технології. Це допомагає знайти робочі місця, що найкраще відповідають потребам та інтересам кандидата.

Однією з основних переваг DOU є його велика та активна спільнота ІТ-спеціалістів. Сайт пропонує форуми та блоги, де фахівці можуть ділитися досвідом, задавати питання та обговорювати актуальні теми з ІТ-сфери. Це створює сприятливе середовище для взаємодії та обміну знаннями між співтовариством.

Також DOU організовує регулярні ІТ-конференції, тренінги та інші події, які сприяють професійному розвитку співтовариства та сприяють побудові мережі контактів.

Ще одним впливовим ресурсом-аналогом є Djinni [2]. Цей сайт пропонує широкий вибір вакансій в різних сферах ІТ, таких як розробка програмного забезпечення, аналітика даних, веб-розробка, тестування та багато іншого.

Один з унікальних аспектів Djinni полягає в можливості анонімного пошуку роботи, що дозволяє користувачам шукати вакансії та додавати резюме, не розкриваючи особисту ідентифікацію.

На Djinni можна знайти розширену інформацію про компанії, яка допомагає кандидатам отримати уявлення про робочі умови, професійний стек та культуру компаній.

Сайт забезпечує зручну систему пошуку, яка дозволяє користувачам фільтрувати вакансії за різними критеріями, такими як розташування, тип роботи, зарплата та інші параметри.

Крім цього, Djinni надає корисні ресурси, такі як блоги та статті, що стосуються кар'єри в ІТ сфері, процесу пошуку роботи, підготовки до

співбесіди та багато іншого. Ці матеріали допомагають користувачам бути кращими кандидатами та розширювати свої знання в галузі ІТ.

Djinni має активну спільноту ІТ-фахівців, яка об'єднує професіоналів з різних галузей ІТ. Ви можете спілкуватися з іншими фахівцями, обмінюватися досвідом, задавати питання та отримувати поради щодо кар'єрного росту та розвитку.

Work.ua [3] є одним з найбільших робочих ресурсів в Україні, який охоплює різні галузі, включаючи ІТ. Він пропонує широкий вибір вакансій в різних регіонах країни.

Work.ua має велику кількість вакансій у сфері ІТ, що дозволяє кандидатам знайти роботу, яка відповідає їхнім потребам та навичкам. Сайт регулярно оновлюється новими вакансіями від різних компаній.

Сайт дозволяє кандидатам фільтрувати вакансії за різними критеріями, такими як місце розташування, тип роботи, рівень досвіду та багато іншого.

Work.ua надає інформацію про компанії, їхні проекти, вимоги до кандидатів та умови праці. Користувачі можуть ознайомитися з детальною інформацією про компанії, переглянути відгуки від співробітників та зробити більш обґрунтоване рішення щодо пошуку роботи.

Таблиця 1.1.

Порівняльна таблиця ресурсів-аналогів

Особливості	DOU	Djinni	Work
Широкий вибір вакансій	+	+	+
Зручний пошук та фільтрація	+	+	+
Професійна спільнота	+	+	-
Можливість анонімного пошуку	-	+	-
Ресурси для розвитку	+	+	-
Актуальна база вакансій	+	+	+

1.3. Вибір архітектури сайту з працевлаштування в сфері ІТ-технологій

Сайт буде побудований згідно архітектури RESTful API [4]. Оскільки RESTful API базується на стандартному протоколі HTTP та використовує формати передачі даних, такі як JSON або XML, розроблений сервер буде сумісний з будь-якою платформою або мовою програмування.

У цій архітектурі додатку, було обрано Symfony [5] як технологію серверної сторони, що реалізує RESTful API, бібліотеку Doctrine [6], що використовується для роботи з базою даних MySQL [7] та клієнтську сторону, реалізовану за допомогою React [8]. Для зручності створення та керування RESTful API буде використано ApiPlatform [9]. Для того щоб забезпечити комунікацію між сервером та клієнтом у реальному часі, буде використаний протокол Mercure [10]. Все це буде розгорнуто на середовищі Docker [11] з використанням Nginx [12] як вебсервера.

Основний принцип роботи цієї архітектури полягає у розділенні функціональності між серверною та клієнтською сторонами додатку. Серверна сторона, реалізована за допомогою Symfony, надає RESTful API для взаємодії з клієнтом.

Symfony дозволяє визначати маршрути, які відповідають різним запитам від клієнта, і виконує відповідні дії. Запити включають HTTP методи, такі як GET, POST, PUT і DELETE, які відповідають операціям з ресурсами. Symfony обробляє ці запити, взаємодіє з базою даних за допомогою Doctrine і повертає відповідь у вигляді JSON.

Клієнтська сторона, реалізована за допомогою React, відображає інтерфейс користувача і взаємодіє з сервером через API. Клієнтська програма React відправляє HTTP запити до сервера, за допомогою бібліотеки Axios [13] для асинхронних запитів, отримує дані та відмальовує актуальний інтерфейс додатку. На рисунку 1.1 зображено схему роботи додатку.

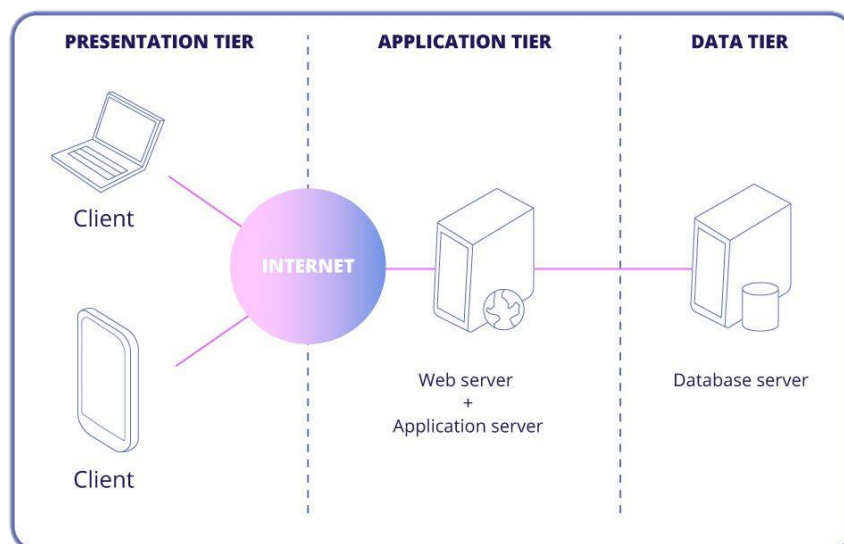


Рисунок 1.1 - Схема роботи клієнт-сервер додатку

Усе це розгорнуто на середовищі Docker, що дозволяє створити контейнеризовані середовища для серверної і клієнтської частин додатку. Контейнеризація дозволяє легко розгортати, масштабувати і керувати додатком.

Nginx використовується як веб-сервер, який служить проксі-сервером для перенаправлення запитів клієнта до відповідного контейнера з Symfony або React. Він також забезпечує обробку статичних файлів та управління HTTP-запитами.

Підчас розробки програмного продукту будуть задіяні наступні патерни:

- **Dependency Injection:** використання патерну Dependency Injection дозволяє зручно управляти залежностями між класами. За допомогою DI контейнера Symfony, можна визначати, внедрювати та управляти залежностями між класами в додатку, що дозволяє забезпечити більшу модульність та перевикористання коду;
- **Repository:** використання патерну Repository в Doctrine дозволяє розділити логіку доступу до даних від моделей даних. Це дозволяє зручно виконувати запити до бази даних та отримувати об'єкти

моделей. В результаті код стає більш модульним і зручним для тестування;

- Active Record: використання патерну Active Record в Doctrine дозволяє зв'язувати об'єкти моделей з записами в базі даних. Це спрощує взаємодію з базою даних, надаючи зручні методи для збереження, оновлення та видалення даних.
- Event-driven Architecture: Symfony також підтримує подієво-орієнтовану архітектуру. Це означає, що під час розробки можна створювати та обробляти події, які виникають в додатку. Це дозволяє виконувати певну логіку при виникненні певних подій і реагувати на них відповідним чином;
- Object-Relational Mapping (ORM): Використання ORM в Doctrine дозволяє зручно взаємодіяти з базою даних. Це означає, що можна працювати з даними у зручному об'єктно-орієнтованому підході, використовуючи звичні методи та запити, замість прямої роботи з SQL-запитами.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Symfony – це потужний фреймворк для розробки веб-додатків на мові PHP. Він надає широкий набір функцій та інструментів, які спрощують процес розробки, забезпечують структуру та організацію коду. В порівнянні з Laravel, він має більш гнучку архітектуру, забезпечуючи широкі можливості налаштування і розширення. Він також відомий своєю високою продуктивністю та надійністю.

Основні переваги використання Symfony:

- стандартизована структура: Symfony пропонує структуру додатку, що дозволяє організувати код згідно з найкращими практиками та стандартами розробки;

- розширюваність: Symfony має модульну архітектуру, яка дозволяє легко додавати та використовувати розширення, бібліотеки та компоненти, що спрощує розширення функціональності вашого додатку
- багатий набір компонентів: Symfony постачається з великим набором готових компонентів, які забезпечують різноманітну функціональність, включаючи обробку HTTP-запитів, роботу з базами даних, кешування, форми, безпеку та багато іншого;
- спільнота та підтримка: Symfony має велику та активну спільноту розробників, яка надає підтримку, документацію та навчальні ресурси, що допомагають вам у процесі розробки.

React – це популярна JavaScript бібліотека для розробки інтерфейсів користувача. Вона дозволяє створювати ефективні, масштабовані та зручні для використання користувацькі інтерфейси. React в порівнянні з іншими бібліотеками, наприклад Vue або Angular, має перевагу у швидкості відображення змін на сторінці, завдяки використанню віртуального DOM і механізму React Hooks.

Основні переваги використання React:

- компонентна архітектура: React базується на компонентній архітектурі, де інтерфейс розбивається на невеликі, повторно використовувані компоненти. Це дозволяє легко організувати код, забезпечити модульність та зручне управління станом додатку;
- віртуальний DOM: React використовує віртуальний DOM, що дозволяє ефективно оновлювати лише необхідні частини інтерфейсу, що призводить до високої продуктивності та швидкодії додатку;
- реактивний підхід: React дозволяє відслідковувати зміни стану та автоматично оновлювати відповідні компоненти, що спрощує управління станом та реагування на події.

Docker – це платформа для контейнеризації додатків, яка дозволяє упаковувати додатки та їх залежності в контейнери, що забезпечує їхню

переносимість та скорочує час на розгортання та налаштування. Порівняно з аналогами, такими як Podman та LXD, Docker має більш значущу перевагу в ефективності ресурсів. Docker пропонує оптимізовану віртуалізацію на рівні контейнерів, що дозволяє забезпечити більш економне використання системних ресурсів, таких як CPU, пам'ять та дисковий простір. Це дозволяє знизити витрати на інфраструктуру та підвищити масштабованість додатків, забезпечуючи більш ефективне використання обчислювальних ресурсів.

Основні переваги використання Docker:

- переносимість: Docker контейнери є переносимими та незалежними від конкретної системи, що дозволяє легко розгорнути додатки на різних середовищах, таких як розробка, тестування та продакшн;
- ізоляція: Контейнеризація дозволяє ізолювати додатки та їхні залежності, що гарантує стабільність та безпеку. Кожен контейнер має своє власне середовище, що запобігає конфліктам та впливу одного додатку на інший;
- швидкість розгортання: Docker дозволяє швидко розгорнути та масштабувати додатки за допомогою контейнерів. Він пропонує інструменти для автоматизації розгортання та оркестрації контейнерів.

Ця архітектура дозволяє розділити серверну та клієнтську частини додатку, що дозволяє розробникам працювати над ними незалежно. Крім того, використання Docker дозволяє забезпечити консистентність середовища розробки та прод середовища, а також спростити розгортання та масштабування додатку.

Загалом використання Symfony, React та Docker дозволяє створити додатки, які є ефективними, масштабованими та зручними у розробці та розгортанні. Ці технології забезпечують швидку відповідь користувачеві та зручний інтерфейс, що дозволяє легко взаємодіяти з додатком.

Висновки до першого розділу

Аналіз напрямків використання, дозволив виявити основні аспекти, які необхідно врахувати при розробці сайту. Підчас вивчення існуючих аналогів було виявлено що найважливішими характеристиками майбутньої системи є широкий вибір та зручність використання.

Після вибору оптимальної архітектури – клієнт-сервер з використанням RESTful API, були обрані підходящі технології для її реалізації, а саме Symfony, React та Docker. В якості аналогів було розглянуто такі технології як Laravel, Vue.js, Angular, Podman та LXD. Проте обрані технології є найбільш підходящими та сучасними для успішної реалізації поставлених завдань.

РОЗДІЛ 2. ПРОЕКТУВАННЯ САЙТУ З ПРАЦЕВЛАШТУВАННЯ У СФЕРІ ІТ-ТЕХНОЛОГІЙ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Бізнес-вимоги

1. Основні цілі: проект створюється з метою надання послуг пошуку роботи в ІТ сфері та сприяння встановленню зв'язку між роботодавцями та кандидатами.
2. Розширена функціональність пошуку: сайт забезпечує розширені можливості пошуку вакансій в ІТ сфері, включаючи фільтрацію за різними критеріями, такими як локація, рівень досвіду, технології, заробітна плата тощо.
3. Профілі фахівців: користувачі мають можливість створити своє резюме, де вони можуть вказати свої навички, досвід роботи, освіту та іншу інформацію, що допоможе роботодавцям зробити обґрунтований вибір кандидата.
4. Комунікація між роботодавцями та кандидатами: сайт надає інструменти для зручної комунікації між роботодавцями та кандидатами, такі як система обміну повідомленнями.

Вимоги користувачів

Зовнішні користувачі:

1. Працівник:

- створення власного детального резюме;
- перегляд актуальної інформації відносно резюме конкурентів;
- перегляд актуальної відносно вакансій та компаній що їх створили;
- можливість відгукуватися на підходящі вакансії;
- можливість вести діалог з представником компанії.

На рисунку 2.1 зображено схему варіантів використання для працівника.

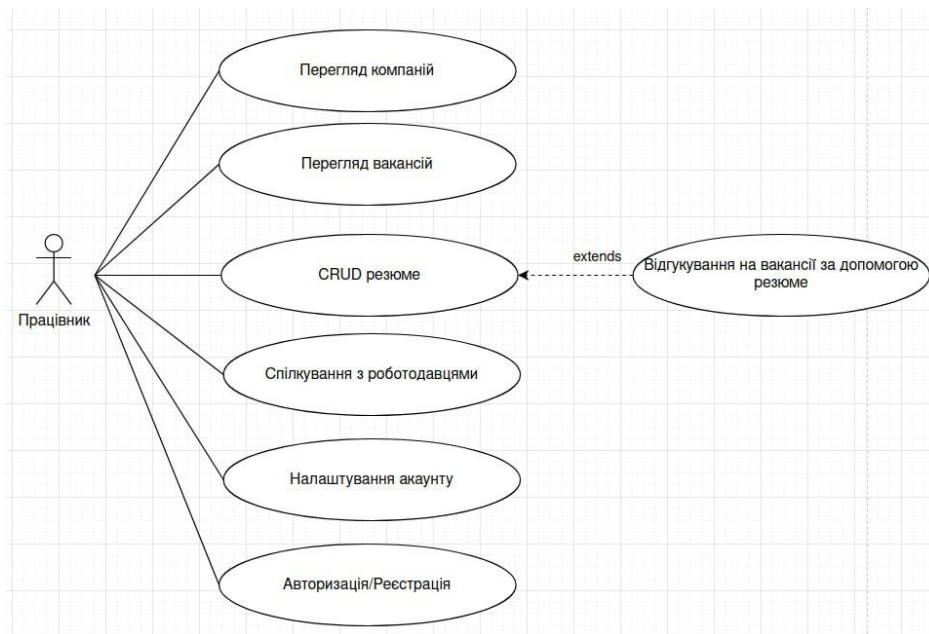


Рисунок 2.1 - Схема варіантів використання для працівника

2. Роботодавець:

- створення компаній з можливістю оплати тарифних планів;
- створення вакансій;
- перегляд актуальної інформації відносно резюме кандидатів;
- перегляд актуальної інформації відносно вакансій та компаній конкурентів;
- створення діалогів з фахівцями.

На рисунку 2.2 зображено схему варіантів використання для роботодавця.

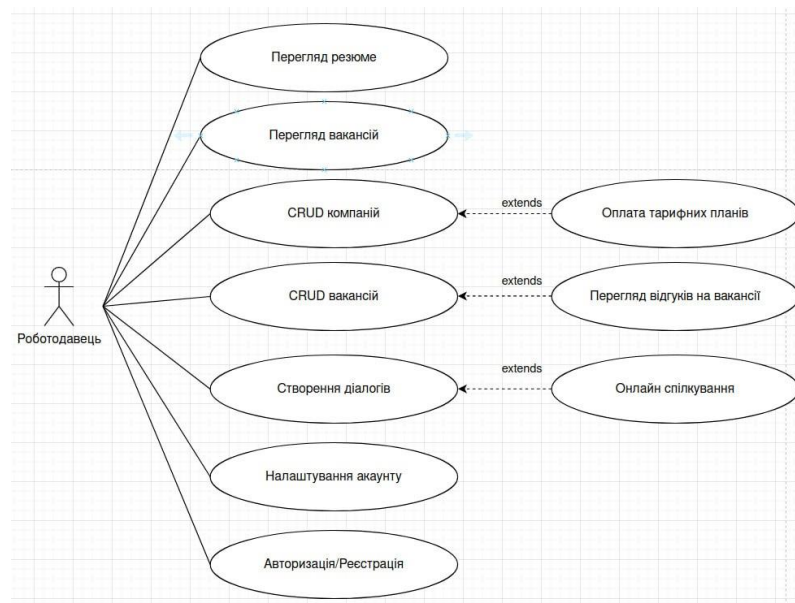


Рисунок 2.2 - Схема варіантів використання для роботодавця

Внутрішній користувач(адміністратор):

- редагування БД
- можливість переглядати компанії, верифікувати або видаляти їх
- можливість переглядати вакансії та резюме
- можливість переглядати транзакції
- можливість переглядати список користувачів

На рисунку 2.3 зображено схему варіантів використання для адміністратора.

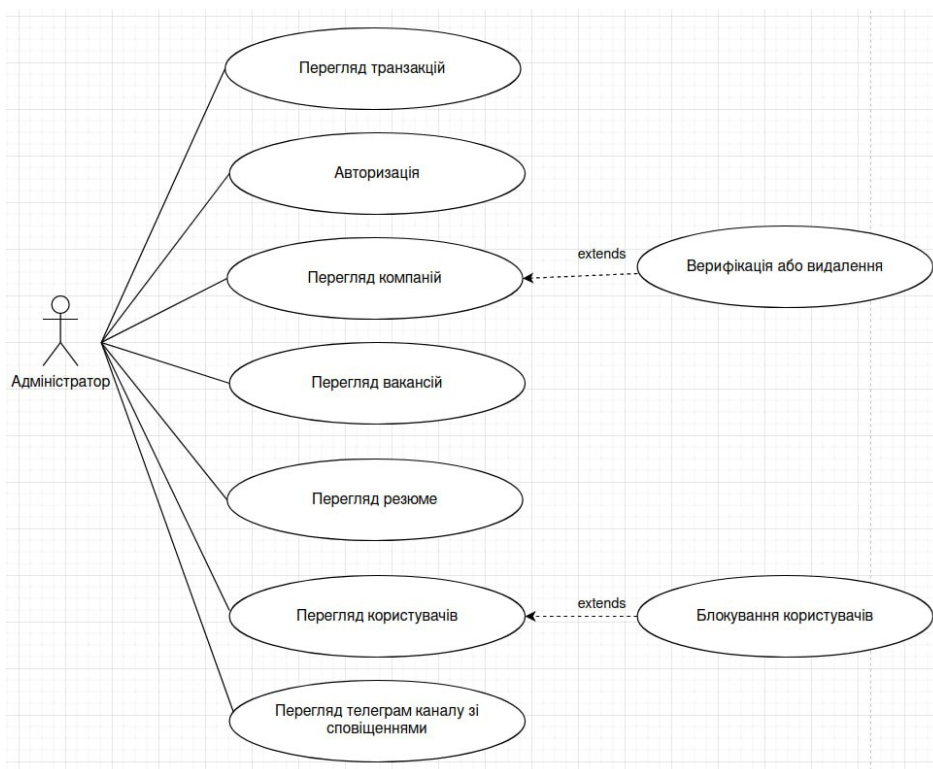


Рисунок 2.3 - Схема варіантів використання для адміністратора

Функціональні вимоги

1. Авторизація та реєстрація користувачів в системі: Повинна бути можливість реєстрація клієнта і автоматичне призначення клієнту певної ролі. Користування функціями сайту потрібно обмежити для неавторизованих користувачів, надавши їм тільки доступ до головної сторінки, сторінки реєстрації та авторизації.

2. Адміністратори сайту та його розробники матимуть доступ до спеціального розділу вебсайту для того щоб мати повну доступ до усієї інформації сайту та мати можливість управляти компаніями.
3. Забезпечення онлайн комунікації: система повинна мати функціонал, що дозволяє роботодавцям взаємодіяти та спілкуватися з кандидатами у режимі реального часу.

Нефункціональні вимоги

1. Час навчання: для звичайних користувачів час, потрібний для ознайомлення з функціоналом сайту та його інструментами, повинен становити 2-3 години, а для досвідчених користувачів цей час може бути скорочений до 1 години.
2. Швидкість відповіді системи: Час відповіді системи для звичайних запитів не повинен перевищувати 3 секунд, а для складних не більше 8 секунд.
3. Доступність: система повинна бути доступною для користувачів протягом більшості часу без перебоїв. Час обслуговування та планові технічні роботи не повинні перевищувати 10% від загального часу роботи сайту.
4. Надійність: сайт повинен забезпечувати безперервну роботу протягом 20 робочих днів без відмов. Максимальна норма помилок та проблем у роботі сайту повинна бути не більше 1 на 1000 запитів користувачів.
5. Навантаження: система повинна витримувати навантаження в 1000 інтенсивно працюючих користувачів та зберігати свою швидкість користування.

Бізнес логіка

1. Сайт є абсолютно безкоштовним для працівників. Вони можуть створювати максимально детальні резюме (по 1 для акаунту) та за допомогою цих резюме відгукуватися на вакансії.
2. Функціонал, представлений для представників компаній, не є безкоштовним. Після створення компаній (максимальна кількість 5)

Продовження табл. 2.1.

email	varchar(255)	—	—	Email користувача
firstname	varchar(255)	—	—	Ім'я користувача
lastname	varchar(255)	—	—	Прізвище користувача
password	varchar(255)	—	—	Хеш паролю акаунту
forgot_password_token	varchar(100)	—	—	Токен для відновлення паролю
email_activation_token	varchar(100)	—	—	Токен для підтвердження акаунту
roles	longtext	—	—	Роль юзера
create_date	int	—	—	Дата реєстрації юзера
is_enabled	tinyint(1)	—	—	Підтвердження акаунту
is_banned	bool	—	—	Блокування акаунту
attempts	int	—	—	Кількість спроб для авторизації
resume_id	char(36)	—	+	Id резюме

В таблиці "user" зберігаються персональні дані про користувача та дані для акаунту(авторизація, активація, зміна паролю та блокування).

Таблиця 2.2.

Структура таблиці "resume"

Назва	Тип даних	ПК	ЗК	Опис поля
id	char(36)	+	—	Id резюме
email	varchar(255)	—	—	Email працівника
firstname	varchar(255)	—	—	Ім'я працівника

Продовження табл. 2.2.

lastname	varchar(32)	—	—	Прізвище працівника
expected_salary	int	—	—	Очікувана зарплатня
experience	double	—	—	Досвід
city	varchar(100)	—	—	Місто працівника
english_level	varchar(100)	—	—	Рівень англійської працівника
job_type	varchar(100)	—	—	Тип бажаної роботи
experience_description	varchar(3200)	—	—	Опис досвіду
achievements_description	varchar(3200)	—	—	Опис досягнень
expectation_description	varchar(3200)	—	—	Опис очікувань
created_at	bigint	—	—	Поле дати створення резюме
updated_at	bigint	—	—	Поле дати останнього оновлення резюме
is_published	tinyint(1)	—	—	Поле публікації
worker_id	char(36)	—	+	Id працівника
portfolio	varchar(100)	—	—	Посилання на портфоліо

В таблиці "resume" зберігаються детальні дані про працівника, його досвід та знання, очікування та побажання.

Таблиця 2.3.

Структура таблиці "company"

Назва	Тип даних	ПК	ЗК	Опис поля
id	char(36)	+	–	Id компанії
employer_id	char(36)	–	+	Id власника компанії
company_plan_id	char(36)	–	+	Id тарифного плану
name	varchar(255)	–	–	Назва компанії
description	longtext	–	–	Опис компанії
is_verified	tinyint(1)	–	–	Верифікація компанії
created_at	bigint	–	–	Поле дати створення компанії
updated_at	bigint	–	–	Поле дати останнього оновлення компанії

В таблиці "company" зберігаються дані про компанії, які створюють роботодавці. Поле "isVerified" служить для того щоб визначити чи компанія є верифікованою адміністрацією сайту.

Таблиця 2.4.

Структура таблиці "transaction"

Назва	Тип даних	ПК	ЗК	Опис поля
id	char(36)	+	–	Id транзакції
company_id	char(36)	–	+	Id компанії
company_plan_id	char(36)	–	+	Id тарифного плану
status	varchar(50)	–	–	Статус оплати
currency	varchar(3)	–	–	Валюта
amount	decimal(6,2)	–	–	Сума
created_at	bigint	–	–	Поле дати створення транзакції
updated_at	bigint	–	–	Поле дати останнього оновлення транзакції

В таблиці "transaction" зберігаються дані про оплату тарифних планів, представниками компаній. Поле "status" призначено для статусу оплати та змінюється тільки внаслідок отримання колбеку від платіжного сервісу LiqPay. Не зважаючи на те, що оплата на сайті буде проводитись тільки у гривнях, було додано поле "currency" для можливості оплати різними валютами у майбутньому.

Таблиця 2.5.

Структура таблиці "company_plan"

Назва	Тип даних	ПК	ЗК	Опис поля
id	int	+	–	Id тарифного плану
name	varchar(255)	–	–	Назва тарифного плану
price	int	–	–	Ціна тарифного плану
priority	int	–	–	Пріоритет тарифного плану
vacancies_amount	varchar(255)	–	–	Кількість можливих створених вакансій

В таблиці "company_plan" зберігаються тарифні плани, які будуть обирати роботодавці для своїх компаній.

Таблиця 2.6.

Структура таблиці "dialog"

Назва	Тип даних	ПК	ЗК	Опис поля
id	char(36)	+	–	Id діалога
company_id	char(36)	–	+	Id компанії
worker_id	char(36)	–	+	Id фахівця
created_at	bigint	–	–	Поле дати створення діалогу
updated_at	bigint	–	–	Поле дати останнього оновлення діалогу
status	varchar(50)	–	–	Статус діалога

Таблиця "dialog" призначена для зберігання діалогів між представниками компаній та спеціалістами. Поле "status" призначено для того щоб представник компанії міг закрити діалог зі спеціалістом.

Таблиця 2.7.

Структура таблиці "message "

Назва	Тип даних	ПК	ЗК	Опис поля
id	char(36)	+	–	Id повідомлення
dialog_id	char(36)	–	+	Id діалога
message	text	–	–	Текст повідомлення
author	varchar(10)	–	–	Автор повідомлення
created_at	bigint	–	–	Поле дати створення повідомлення
updated_at	bigint	–	–	Поле дати останнього оновлення повідомлення

В таблиці "message" зберігаються дані повідомлення з діалогу.

Таблиця 2.8.

Структура таблиці "vacancy_resume"

Назва	Тип даних	ПК	ЗК	Опис поля
id	char(36)	+	–	Id відгуку на вакансію
vacancy_id	char(36)	–	+	Id вакансії
resume_id	text	–	+	Id резюме працівника
created_at	bigint	–	–	Поле дати створення відгука
updated_at	bigint	–	–	Поле дати останнього оновлення відгука

Таблиця "vacancy_resume" призначена для зберігання відгуків працівників на вакансії.

Таблиця 2.9.

Структура таблиці "company_contacts"

Назва	Тип даних	ПК	ЗК	Опис поля
id	int	+	–	Id контакту в компанії
company_id	char(36)	–	+	Id компанії
contacts_id	int	–	+	Id контакту
value	varchar(255)	–	–	Значення контакту

Таблиця "company_contacts" призначена для зберігання контактної інформації компанії.

Таблиця 2.10.

Структура таблиці "resume_technologies"

Назва	Тип даних	ПК	ЗК	Опис поля
id	int	+	–	Id технології у резюме
resume_id	char(36)	–	+	Id резюме
value	varchar(255)	–	–	Назва технології

Таблиця "resume_technologies" призначена для зберігання технологій з якими працівник має досвід.

Таблиця 2.11.

Структура таблиць "technologies", "city", "category"

Назва	Тип даних	ПК	ЗК	Опис поля
id	char(36)	+	–	Id технології
name	varchar(255)	–	–	Назва технології

Ці таблиці призначені для того, щоб містити у собі загальну інформацію яка потім буде використана при створенні компаній та резюме.

Таблиця 2.12.

Структура таблиці "media_object"

Назва	Тип даних	ПК	ЗК	Опис поля
id	char(36)	+	–	Id медіа файлу
company_logo_id	char(36)	–	+	Id компанії за полем лого
company_document_id	char(36)	–	+	Id компанії за полем документ
content_url	varchar(150)	–	–	Назва медіа файлу
storage	varchar(50)	–	–	Назва папки медіа файлу
type	varchar(25)	–	–	Тип медіа файлу
resume_photo_id	char(36)	–	+	Id резюме за полем фото
resume_cv_id	char(36)	–	+	Id резюме за полем cv

Таблиця "media_object" призначена для зберігання шляху на сервері до медіа файлів.

Таблиця 2.13.

Структура таблиці "settings"

Назва	Тип даних	ПК	ЗК	Опис поля
id	int	+	–	Id налаштування
name	varchar(255)	–	–	Назва налаштування
type	varchar(255)	–	–	Тип налаштування

Таблиця 2.14.

Структура таблиці "user_settings"

Назва	Тип даних	ПК	ЗК	Опис поля
user_id	char(36)	+	–	Id користувача
settings_id	int	–	+	Id налаштування
value	varchar(255)	–	–	Значення налаштування

Таблиці "settings" та "user_settings" призначені для налаштування акаунту, наприклад двоетапної авторизації.

Таблиця 2.15.

Структура таблиці "contacts"

Назва	Тип даних	ПК	ЗК	Опис поля
id	int	+	–	Id контакту
name	varchar(255)	–	–	Назва контакту
type	varchar(255)	–	–	Тип контакту

Таблиця "contacts" призначена для зберігання типів контактної інформації яка може бути використана при створенні резюме чи компанії.

Таблиця 2.16.

Структура таблиці "resume_contacts"

Назва	Тип даних	ПК	ЗК	Опис поля
id	int	+	–	Id контакту резюме

resume_id	char(36)	—	—	Id резюме
contacts_id	int	—	—	Id контакту
value	varchar(255)			Значення контакту резюме

Таблиця "resume_contacts" призначена для зберігання контактної інформації резюме.

2.3. Проектування та реалізація алгоритмів роботи системи

Для роботи з додатком користувач повинен бути зареєстрованим у системі. На сайті передбачено реєстрація за допомогою Google-акаунту та шляхом заповнення форми. Другий варіант передбачає активацію акаунта, шляхом переходу по посиланню, яке прийде на пошту.

Після реєстрації користувач отримує відразу роль працівника. Тепер він може заповнити своє резюме та приступати до пошуку роботи. Проте якщо користувач має намір шукати фахівців для своїх компаній, йому потрібно буде перейти в налаштування та у відповідному розділі, скористатись функціоналом зміни ролі. Уся його діяльність, як працівника, наприклад резюме, відгуки чи діалоги, буде видалена. Відразу після цього, йому буде доступно створення компаній. При створенні компанії, обов'язковою умовою є завантаження документу, що підтвердить її легітимність. Перелік документів буде надано користувачеві. Після створення компанії, адміністратор отримає сповіщення у телеграм-канал, що дозволить йому оперативно перевірити документи компанії та верифікувати компанію на сайті, або ж зв'язатись по пошті з представником компанії для прохання завантажити інший документ. Допоки компанія не буде верифікована, користувач матиме змогу завантажувати інші документи, після чого адміністратор буде також отримувати сповіщення у телеграм-канал.

Для функціонування верифікованих компаній, на сайті буде обов'язковим покупка тарифних планів для них. З усіма тарифними планами користувач зможе ознайомитись на відповідній сторінці та вибрати підходящий саме для

його компанії. Процес оплати буде проводитись за допомогою платіжної системи LiqPay [15]. Процес оплати зображено на рисунку 2.5.

Перш за все, клієнт обере компанію, для якої хоче придбати тарифний план. Далі користувач натисне кнопку "Оплатити", після чого його переадресує на сторінку оплати сервісу LiqPay. На цій сторінці буде зазначена сума оплати та платіжний метод. Користувачеві потрібно буде коректно заповнити платіжні дані своєї картки та дочекатись повідомлення про успішну оплату. Після чого користувач матиме змогу повернутись назад на сайт та після отримання колбеку від LiqPay побачити тарифний план у відповідній компанії.

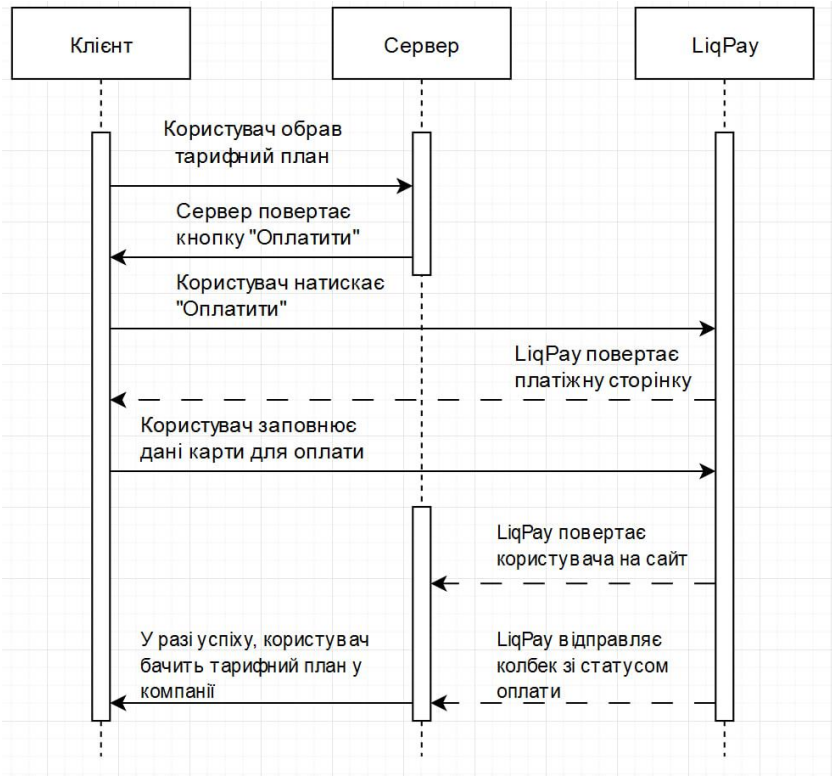


Рисунок 2.5 - Діаграма послідовності процесу оплати

Після отримання тарифного плану, компанія зможе публікувати власні вакансії. В деталях вакансії, представник буде мати змогу бачити фахівців, які відгукнулися на неї. В деталях резюме фахівців, представник компанії матиме змогу створити діалог зі спеціалістом. Оскільки у роботодавця передбачена наявність декількох компаній, перед початком діалогу, він зазначить від лиця якої компанії він буде спілкуватись з фахівцем. Після завершення діалогу, представник компанії матиме змогу закрити даний діалог. Це заборонить

спеціалісту писати в цей діалог. Проте по бажанню роботодавця, він зможе відновити даний діалог та продовжити спілкування.

Для чату буде використовуватись Mercure Hub, оскільки це технологія, яка забезпечує комунікацію між сервером і клієнтами у реальному часі. Це дозволить користувачам отримувати оновлення повідомлень миттєво без необхідності постійного оновлення сторінки. Таким чином, чат буде миттєвим та інтерактивним, дозволяючи користувачам обмінюватись повідомленнями у реальному часі без затримок. Процес спілкування зображено на рисунку 2.6.

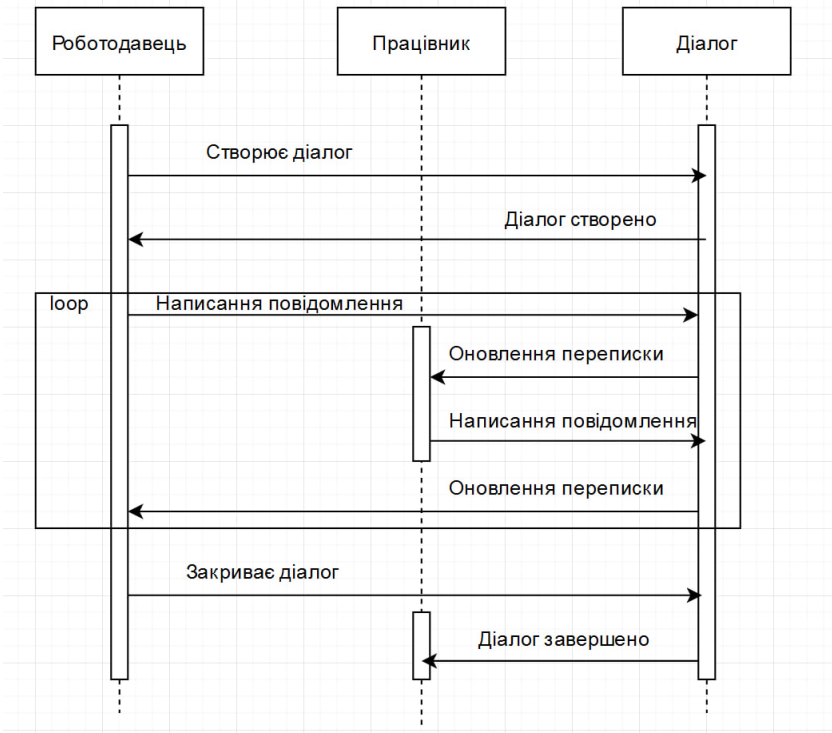


Рисунок 2.6 - Діаграма послідовності процесу спілкування

2.4. Реалізація сайту працевлаштування у сфері ІТ-технологій

Одним з основних модулів додатку є модуль резюме. Для можливості створення резюме, в сутності Resume, було додано маршрут з методом POST. Для нього було додано параметри безпеки, групи полів для денормалізації та контролер який буде оброблювати даний запит. На рисунку 2.7 зображено код реалізації створення маршруту з методом POST для створення резюме.

```
'post' => [
    'method' => "POST",
    'denormalization_context' => ['groups' => ['post:collection:resume']],
    'normalization_context' => ['groups' => ['get:item:resume']],
    'controller' => CreateResumeAction::class,
    'security' => "is_granted('ROLE_WORKER')",
],
```

Рисунок 2.7 - Код створення маршруту

На рисунку 2.8 зображено код реалізації створення резюме.

```
/**
 * @param Resume $data
 * @param Request $request
 *
 * @return Resume
 */
public function __invoke(Resume $data, Request $request): Resume
{
    $content = json_decode($request->getContent(), associative: true);

    $user = $this->security->getUser();

    $data->setWorker($user);

    if (isset($content["contacts"])) {
        if (count($content["contacts"]) > self::MAX_CONTACTS) {
            throw new BadRequestHttpException("Максимальна кількість контактів: " . self::MAX_CONTACTS);
        }

        $this->createContacts($data, $content["contacts"]);
    }

    if (isset($content["technologies"])) {
        $this->createResumeTechnologies($data, $content["technologies"]);
    }

    $this->createResumeMediaObjects($data, $content);

    return $data;
}
```

Рисунок 2.8 - Код створення резюме

Для створення резюме, на полях сутності ставляться групи, для того щоб Symfony розумів як денормалізувати об'єкт, що прийшов у запиті, та створити з цих полів об'єкт сутності. Це дає нам змогу мати вже створений об'єкт Resume на початку методу invoke. Для того щоб користувачі не могли створювати резюме від лиця чужих акаунтів, власник резюме визначається за допомогою модуля Security, що дозволяє витягнути поточного користувача. Оскільки зазначення контактів та технологій, які знає користувач, не є обов'язковим,

перевіряється їх наявність у тілі запиту. У разі їх наявності, викликаються методи їх створення. В кінці методу викликається функція для збереження медіа файлів, які додані до резюме. Для того щоб не порушувати принципу Single Responsibility [16], логіка створення контактів, технологій та медіа файлів була винесена в окремі методи.

Для реалізації оплати, було використано модуль LiqPay. Логіка створення оплати наведена на рисунку 2.9.

```
$liqpay = new LiqPay($_ENV["LIQPAY_PUBLIC_KEY"], $_ENV["LIQPAY_SECRET_KEY"]);

$params = array(
    'action'      => 'pay',
    'amount'      => $data->getAmount(),
    'currency'    => $data->getCurrency(),
    'description' => 'Оплати тарифного плану',
    'order_id'    => $data->getId(),
    'version'     => '3',
    'public_key'  => $_ENV["LIQPAY_PUBLIC_KEY"],
    'result_url'  => $_ENV['BASE_URL'] . "company-details/" . $company->getId(),
    'server_url'  => $_ENV['BASE_URL'] . "api/payment-callback",
);

$data = base64_encode(json_encode($params));
$signature = $liqpay->cnb_signature($params);

return new JsonResponse(
    [
        "data"      => $data,
        "signature" => $signature
    ], status: Response::HTTP_OK
);
```

Рисунок 2.9 - Код створення оплати LiqPay

Для створення оплати, було створено об'єкт класу LiqPay, в конструктор якого було передано ключі для підключення. В цілях безпеки, ці ключі було винесено у файл конфігурації середовища. Для створення оплати на сервісі LiqPay, потрібно вказати такі дані як сума, валюта, ідентифікатор транзакції, публічний ключ та посилання для колбеку та переадресації клієнта у разі успішної оплати. На основі цих даних та вищесказаних ключів підключення, генеруються дані для оплати та створюється ключ ідентифікації для підтримки

безпеки платежів. На основі цих даних, користувач отримає кнопку для оплати, яка приведе його на платіжну сторінку сервісу LiqPay.

Оскільки в даному додатку важлива зручність пошуку, було розроблено фільтрацію за багатьма критеріями. На рисунку 2.10 зображено код фільтрації, яка створена за допомогою модуля фільтрації ApiPlatform.

```
#[ApiFilter(SearchFilter::class, properties: [
    'firstname'    => 'partial',
    'lastname'     => 'partial',
    'worker.id'    => 'exact',
    'category'     => 'exact',
    'city'         => 'exact',
    'englishLevel' => 'exact',
    'jobType'      => 'exact'
])]
#[ApiFilter(RangeFilter::class, properties: [
    'experience',
    'expectedSalary'
])]
#[ApiFilter(BooleanFilter::class, properties: [
    'isPublished'
])]
```

Рисунок 2.10 - Реалізація фільтрації

Висновки до другого розділу

Вебдодаток реалізує функціонал працевлаштування та пошуку фахівців в сфері IT-технологій. Для оптимального функціонування додатку, було розроблено детальну структуру бази даних з можливістю подальшого розширення та масштабування.

У розділі детально вивчалися вимоги до системи, завдяки чому було розроблено алгоритми роботи додатку та візуалізовано їх в якості діаграм. В останньому пункті було розглянуто аспекти реалізації важливих модулів системи.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З САЙТОМ ПРАЦЕВЛАШТУВАННЯ У СФЕРІ ІТ-ТЕХНОЛОГІЙ

3.1. Порядок встановлення та налаштування параметрів системи

Мінімальні вимоги:

1. Вимоги до апаратного забезпечення: сервер має задовольняти наступні характеристики:
 - 2ГБ оперативної пам'яті;
 - процесор з частотою не менше 2.4 ГГц;
 - операційна система Linux дистрибутив Ubuntu 18.04 або вище
2. Вимоги до СУБД:
 - У ядрі системи повинна бути представлена СУБД реляційного доступу

Порядок встановлення:

1. Встановити платформу контейнеризації Docker та інструмент для керування багатоконтейнерними додатками Docker Compose [17].
2. Зклонувати з репозиторію проект, перенести в корінь проекту файл з environment конфігурацією з папки environments_data. В цьому файлі знаходяться ключі до SMTP сервера, Google ключі, JWT [18] ключі для Mercure та ключі для LiqPay оплати.
3. Далі потрібно запустити команду prod_recrate_docker.sh. Ця команда запустить контейнери додатку в prod режимі. Підчас виконання скрипту потрібно ввести згенеровані JWT ключі для авторизації.
4. Після виконання скрипта потрібно виконати команду docker ps -a, щоб переконатись що усі контейнери підійняті.

3.2. Структура інтерфейсу сайту з працевлаштування у сфері ІТ-технологій

За допомогою форми реєстрації користувачі матимуть змогу отримати

акаунт працівника. Для цього потрібно буде заповнити поля з персональними даними або ж скористатися можливістю зареєструватися за допомогою Google-акаунта. Якщо ж користувач має на меті створити акаунт саме роботодавця, він зможе його змінити згодом у налаштуваннях акаунта. У випадку якщо користувач зареєструється саме за допомогою форми, йому прийде лист на пошту з підтвердженням та йому потрібно буде перейти за посиланням для активації акаунту.

На рисунку 3.1 зображено форму реєстрації на сайті.

Рисунок 3.1 - Сторінка реєстрації

Форма авторизації також передбачає два варіанти авторизації: за допомогою введення своєї пошти та паролю та за допомогою Google-акаунта. Якщо в користувача ввімкнена двоетапна аутентифікація, наступним кроком буде введення коду. На рисунку 3.2 зображено форма авторизації на сайті.

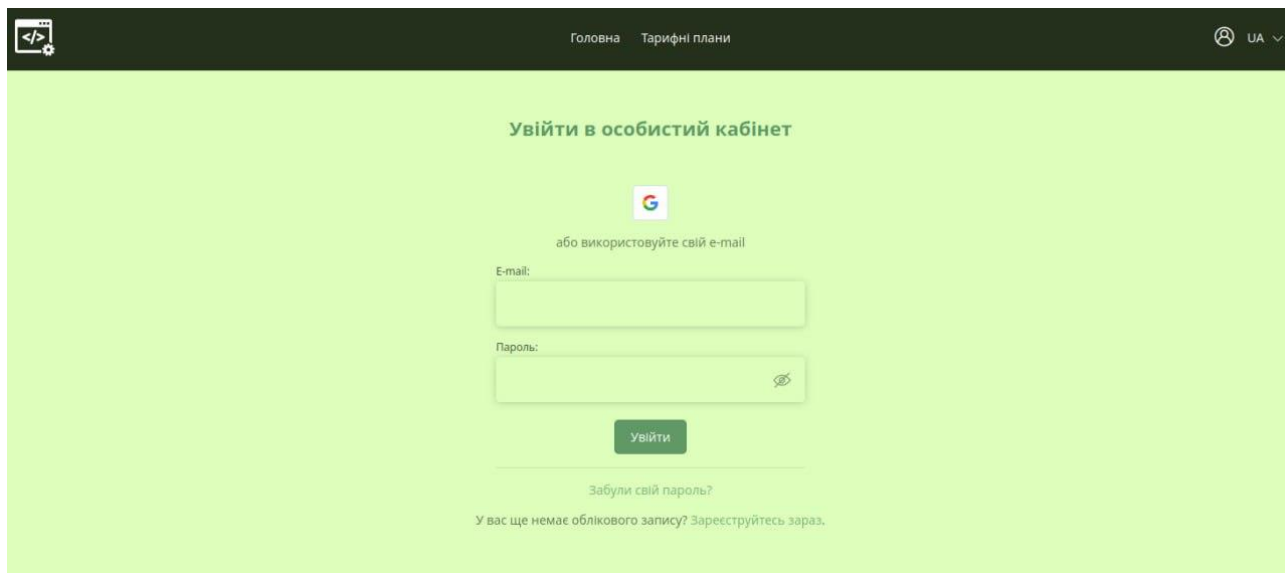


Рисунок 3.2 - Сторінка авторизації

В налаштуваннях акаунту працівники матимуть змогу ввімкнути двоетапну авторизацію, змінити пароль чи стати роботодавцем. Налаштування двоетапної авторизації та зміни паролю також доступне для роботодавців.

На рисунку 3.3 зображено сторінку ввімкнення двоетапної авторизації.

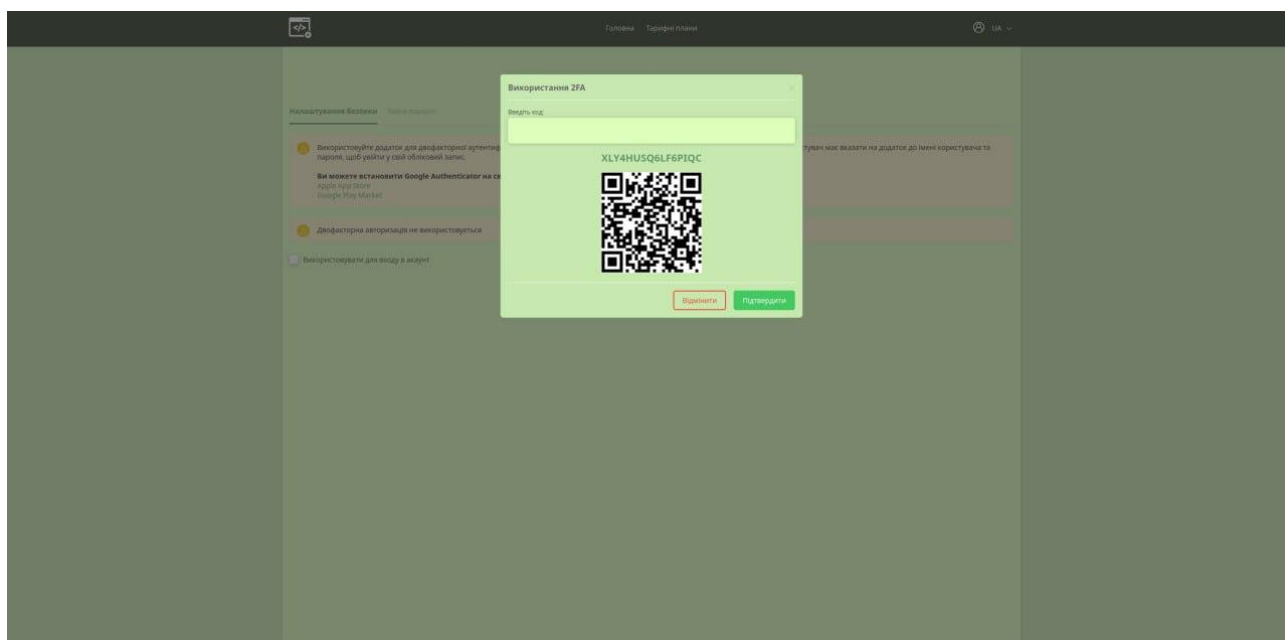


Рисунок 3.3 - Сторінка ввімкнення двоетапної авторизації

На рисунку 3.4 зображено сторінку зміни паролю в налаштуваннях акаунта.

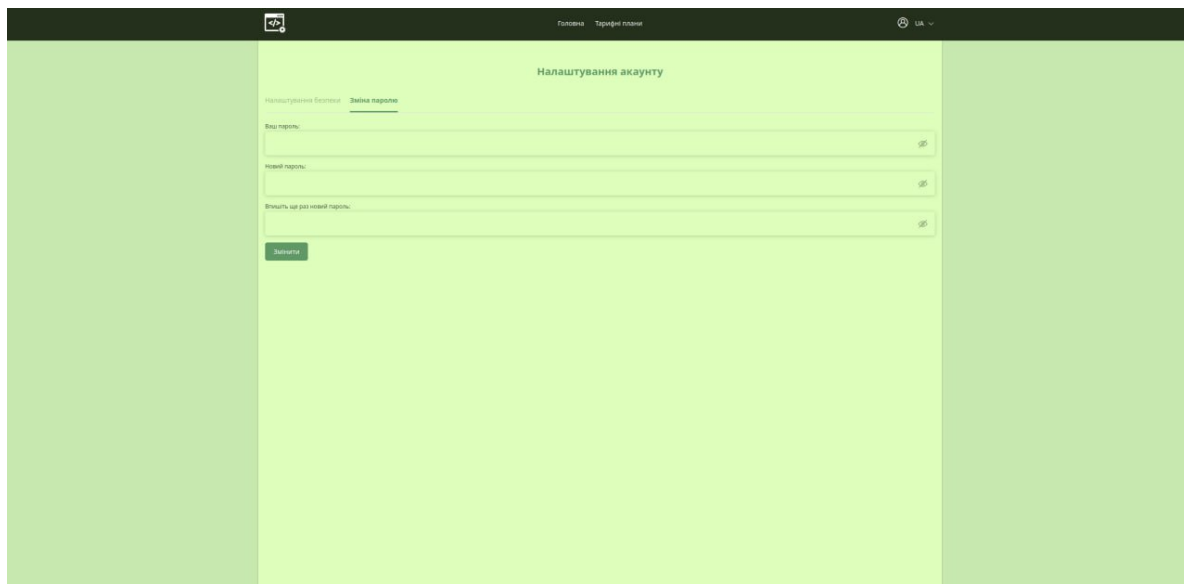


Рисунок 3.4 - Сторінка зміни паролю

Працівник може перейти в розділ резюме. Там йому буде доступно дві вкладки: перша для створення а потім редагування власного резюме, друга для перегляду списку резюме конкурентів з можливістю фільтрації. На сторінці створення чи редагування резюме, користувачеві буде запропоновано надати детальну інформацію за багатьма критеріями, не усі з яких є обов'язковими. При завантаженні власного CV-файлу, в усіх користувачів буде можливість його завантажити або ж переглянути на сторінці. На рисунках 3.5, 3.6 та 3.7 зображено сторінку створення та подальшого редагування резюме.

Рисунок 3.5 - Сторінка створення та редагування резюме

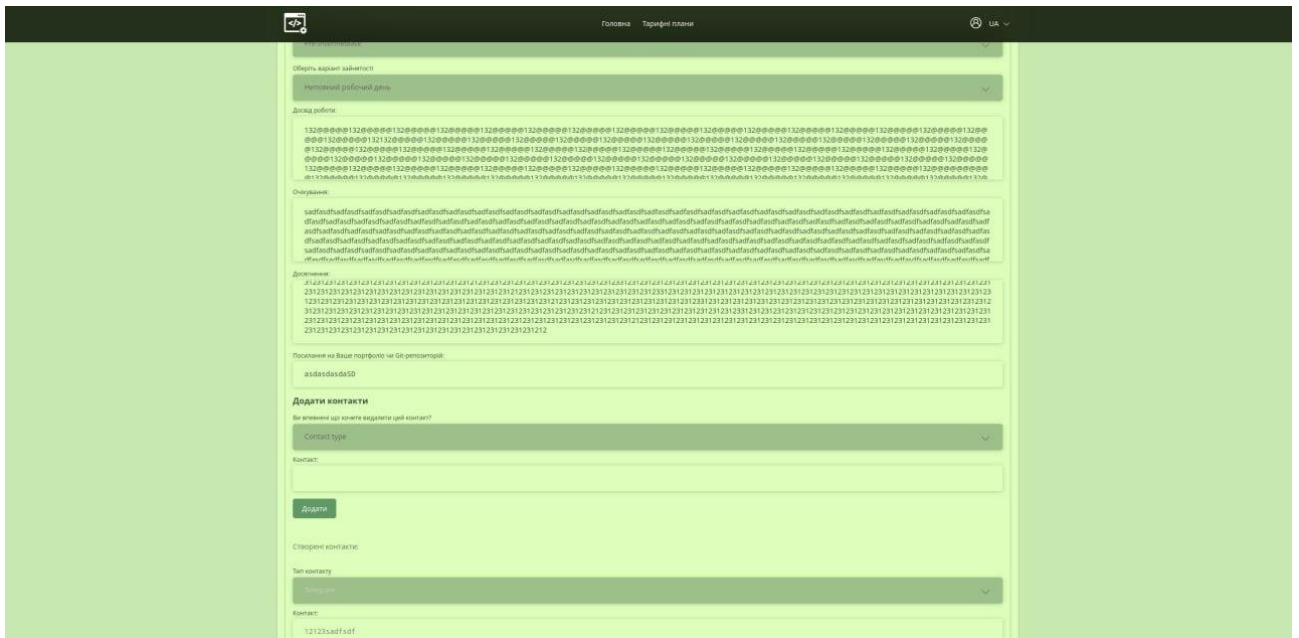


Рисунок 3.6 - Сторінка створення та редагування резюме

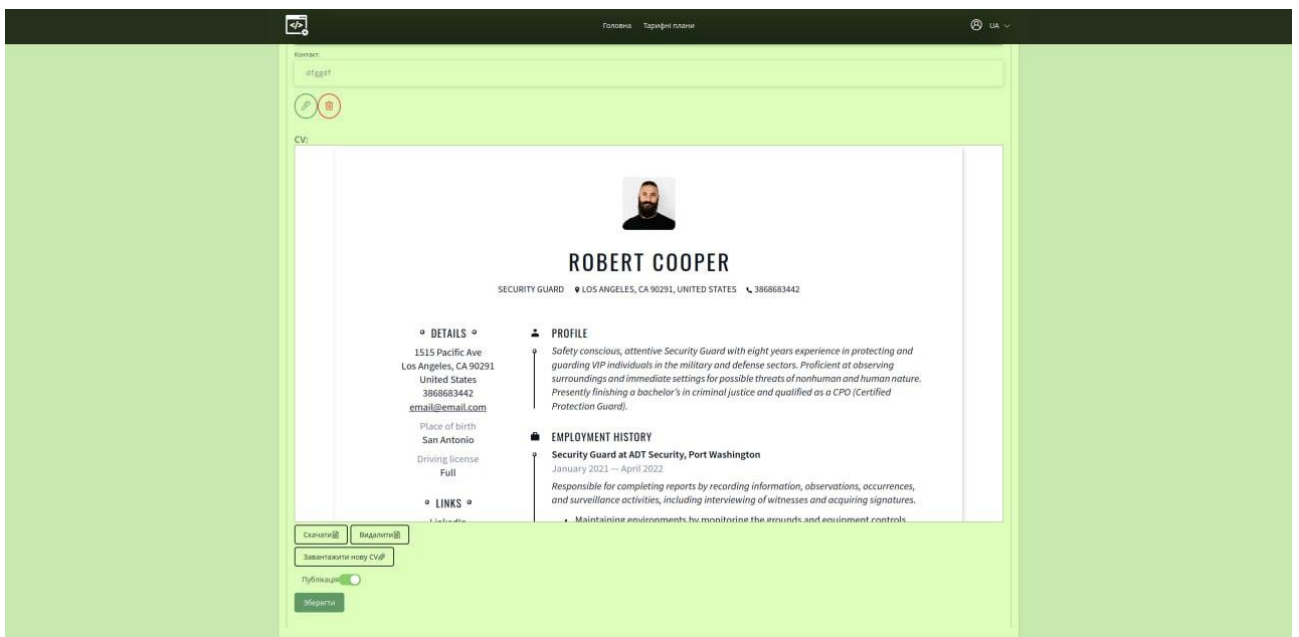


Рисунок 3.7 - Сторінка створення та редагування резюме

Для роботодавців доступні розділи управління компаніями та вакансіями. Для створення компанії, користувачеві буде запропоновано заповнити назву компанії, невеликий опис, створення контактної інформації, завантаження логотипу та завантаження документа в обов'язковому вигляді. На рисунках 3.8 та 3.9 зображено сторінку створення компанії.

Рисунок 3.8 - Сторінка створення компанії

Рисунок 3.9 - Сторінка створення компанії

Для створення вакансій користувачеві запропоновано вказати назву, місто так категорію. Усі інші деталі та потреби користувача, він зможе помістити в опис, який він може створювати за допомогою редактора тексту.

На рисунку 3.10 зображено сторінку створення нової вакансії.

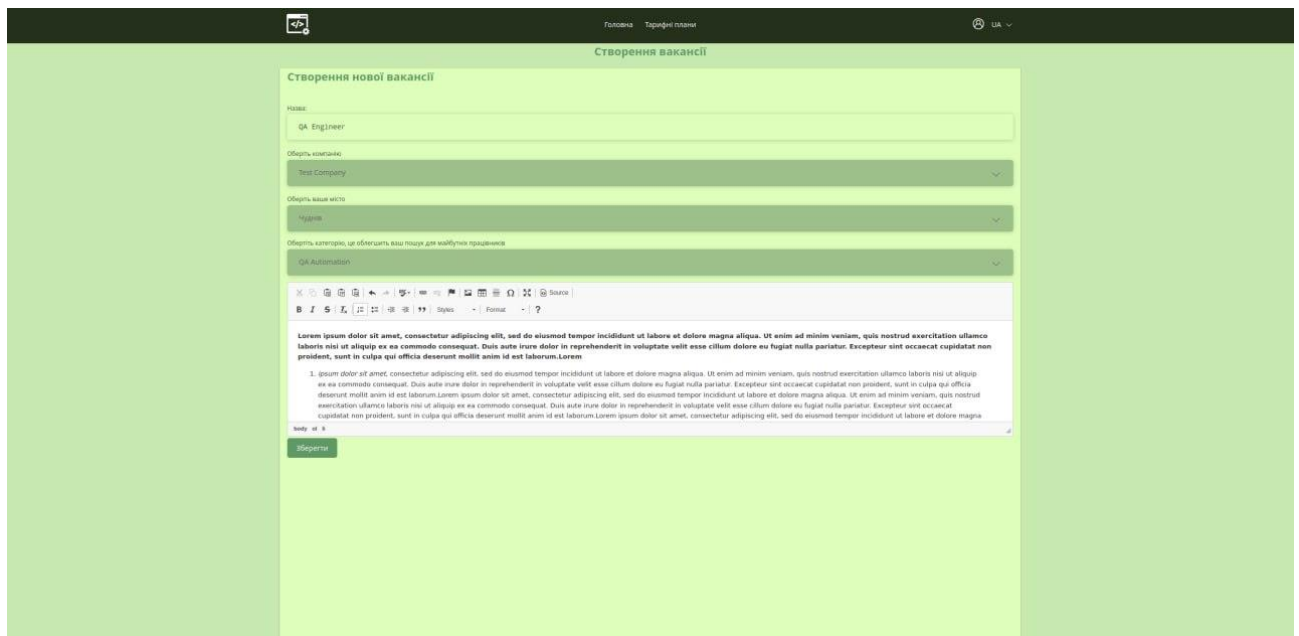


Рисунок 3.10 - Сторінка створення вакансії

Інформація про компанії, вакансії та резюме є публічною, тому доступна для всіх авторизованих користувачів. Для зручності, при перегляді сторінки компанії додано можливість перейти на її вакансії.

На рисунку 3.11 зображено сторінку перегляду компанії з можливістю перейти на список її вакансій.



Рисунок 3.11 - Сторінка перегляду компанії

На рисунку 3.12 зображено сторінку перегляду вакансії.

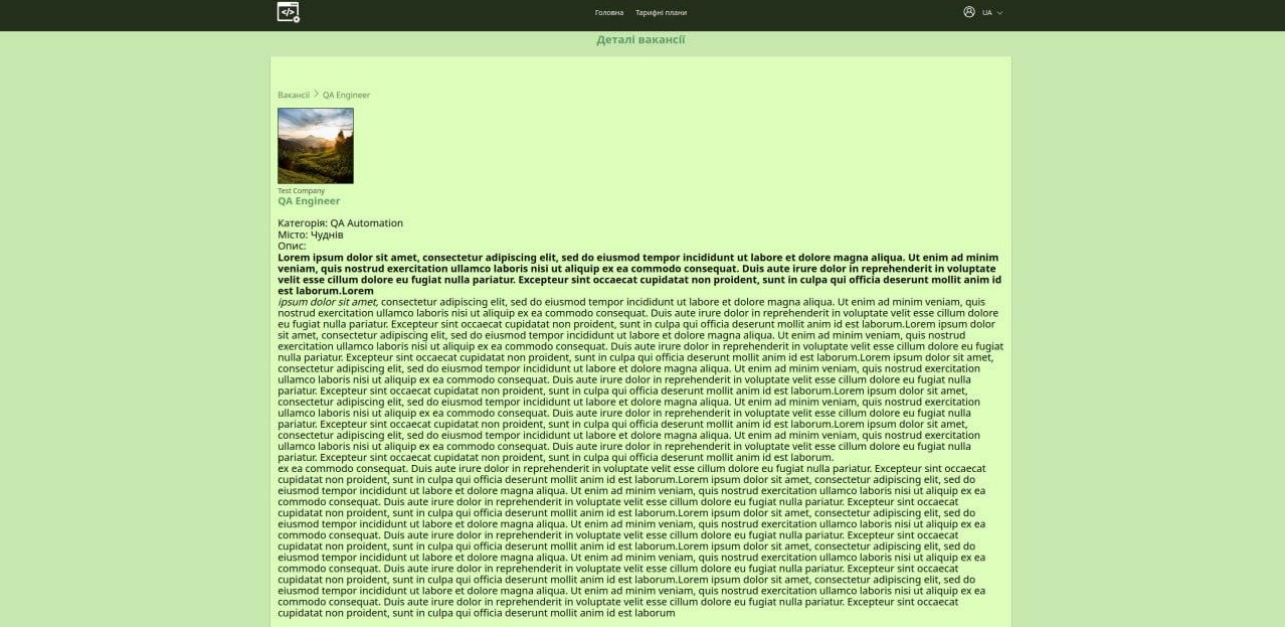


Рисунок 3.12 - Сторінка перегляду вакансії

На рисунках 3.13 та 3.14 зображено сторінку перегляду резюме.

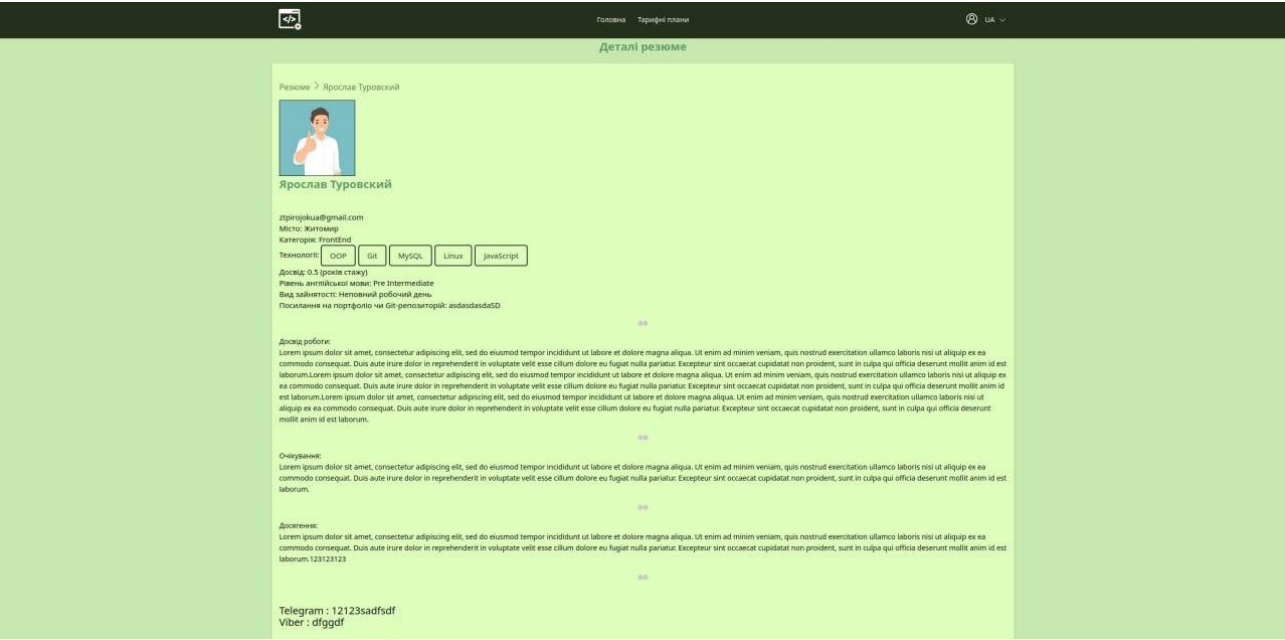


Рисунок 3.13 - Сторінка перегляду резюме

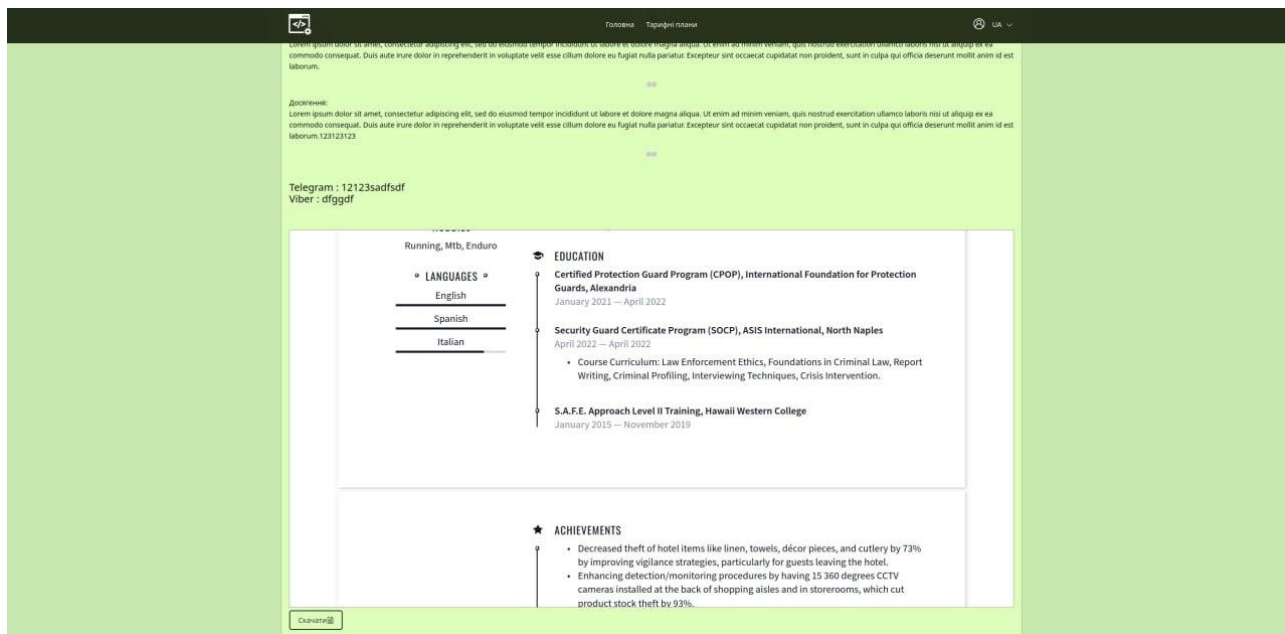


Рисунок 3.14 - Сторінка перегляду резюме

Після того як роботодавці запевняються що фахівець для них підходить, вони мають змогу створити з ним діалог для подальшого спілкування та обговорення умов співбесіди. Оскільки у роботодавця передбачено наявність декількох компаній, він може обрати від лиця якої з компаній він починатиме діалог. На рисунку 3.15 зображено форму для відкриття діалогу.

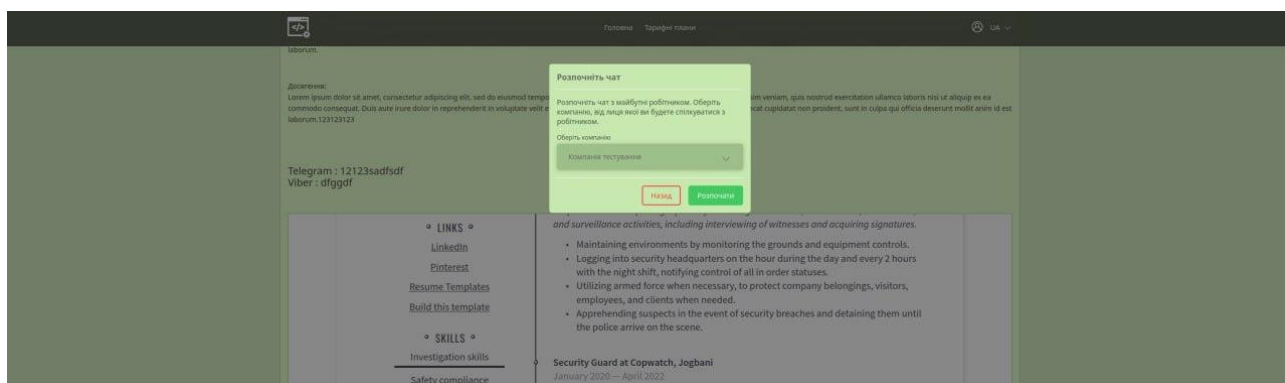


Рисунок 3.15 - Форма для відкриття діалогу з фахівцем з вибором компанії

Чат працює без перезавантажень сторінок, тому є інтерактивним та зручним. Після завершення спілкування, роботодавець має змогу завершити діалог, що заборонить більше писати в нього. При бажанні він зможе його відновити. На рисунку 3.16 зображено сторінку переписки роботодавця та працівника.

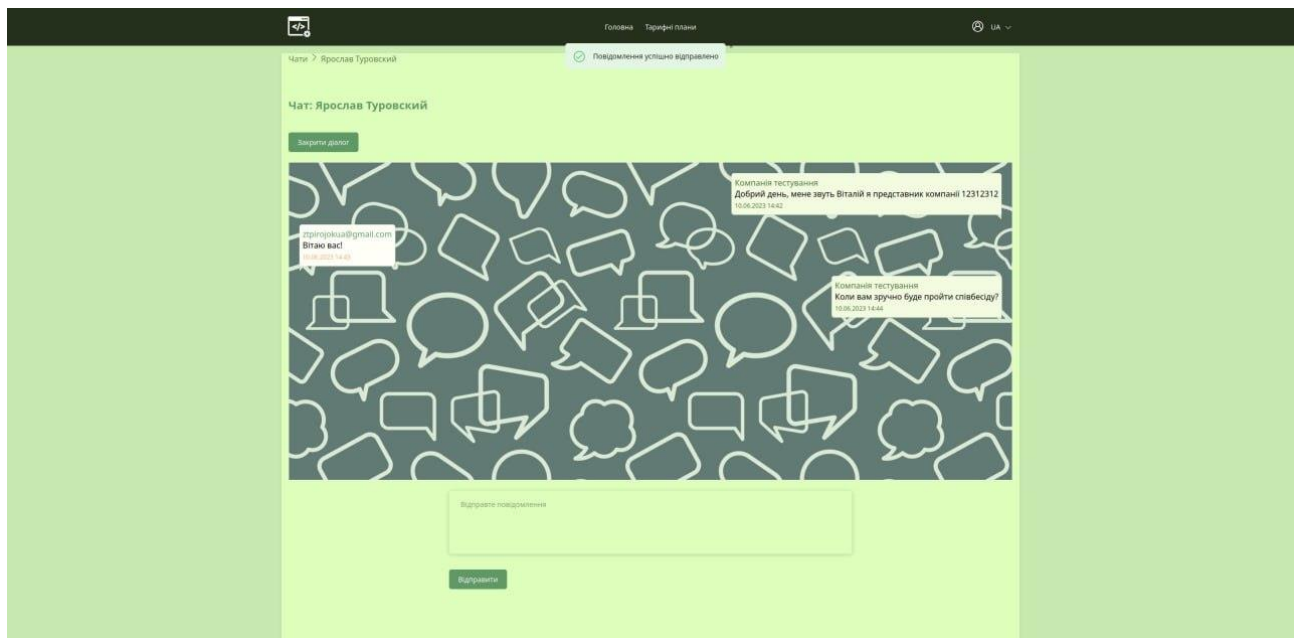


Рисунок 3.16 - Сторінка діалогу роботодавця з працівником

Оскільки важливим аспектом пошуку вакансій, компаній чи резюме є фільтрація, було розроблено фільтра за багатьма критеріями. Для пошуку резюме можливо фільтрувати за іменем, прізвищем, рівнем англійської мови, категорією, містом, видом зайнятості, досвідом та очікуваною зарплатнею.

На рисунку 3.17 зображено сторінку перегляду списку резюме з можливістю фільтрації.

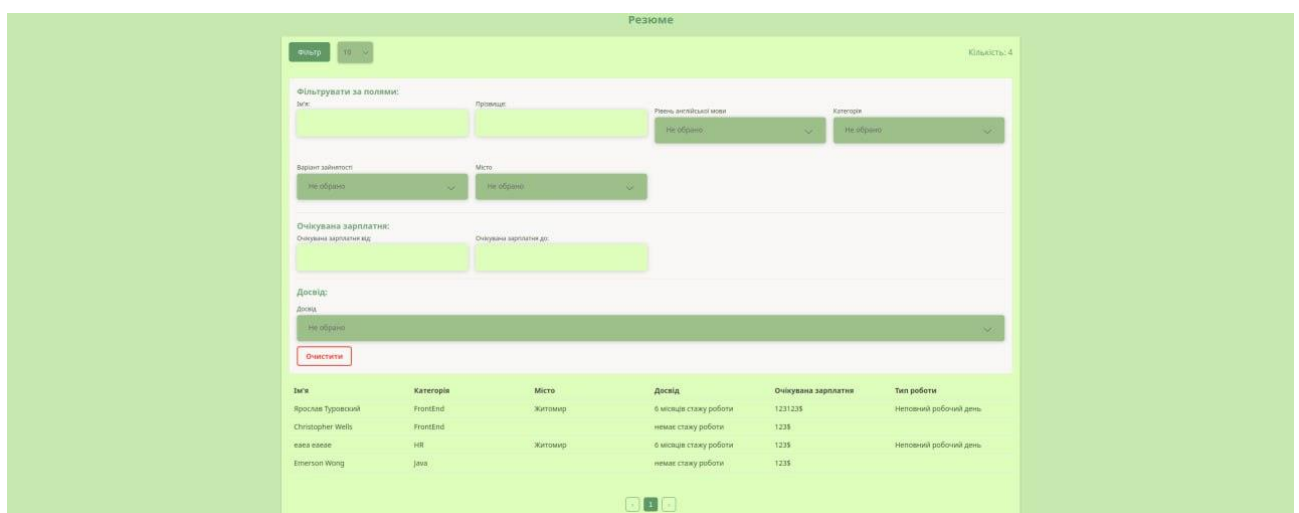


Рисунок 3.17 - Сторінка перегляду списку резюме з фільтрацією з ролі роботодавця

Головна

Тарифні плани

UA

Вакансії

Фільтр

10

Кількість: 3

Фільтрувати за полями:

Назва

Назва компанії

Текст вакансії

Категорія

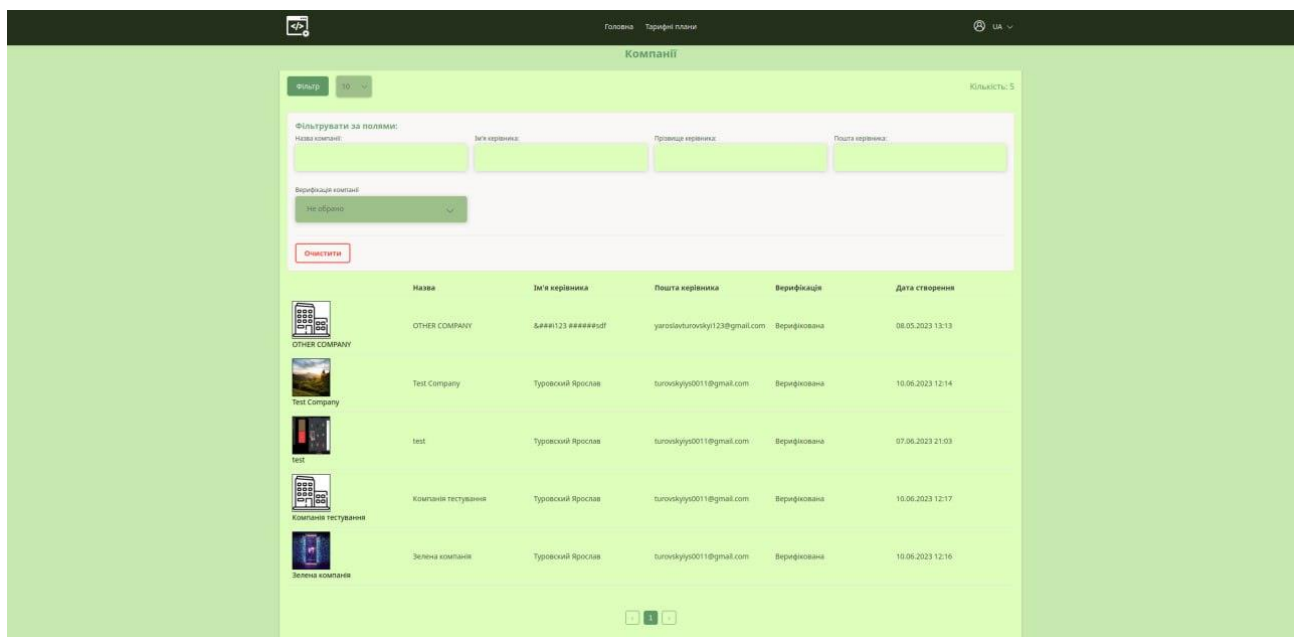
Місто

Очистити

Назва	Компанія	Категорія	Дата створення
QA Engineer	Test Company	QA Automation	10.06.2023 14:21
PHP JUNIOR	Test Company	PHP	10.06.2023 14:24
test	Test Company	QA Manual	10.06.2023 14:24

1

Для зручності пошуку компаній для адміністратора було додано фільтрування за назвою компанії, ім'ям та прізвищем керівника, поштою керівника та статусом верифікації. На рисунку 3.19 зображено сторінку перегляду списку компаній з можливістю фільтрації.



ПЗ.КР.Б – 121 – 23 – ПЗ

Для функціонування компанії, для неї потрібно вибрати тарифний план та придбати його. На рисунку 3.20 зображено сторінку перегляду тарифних планів.

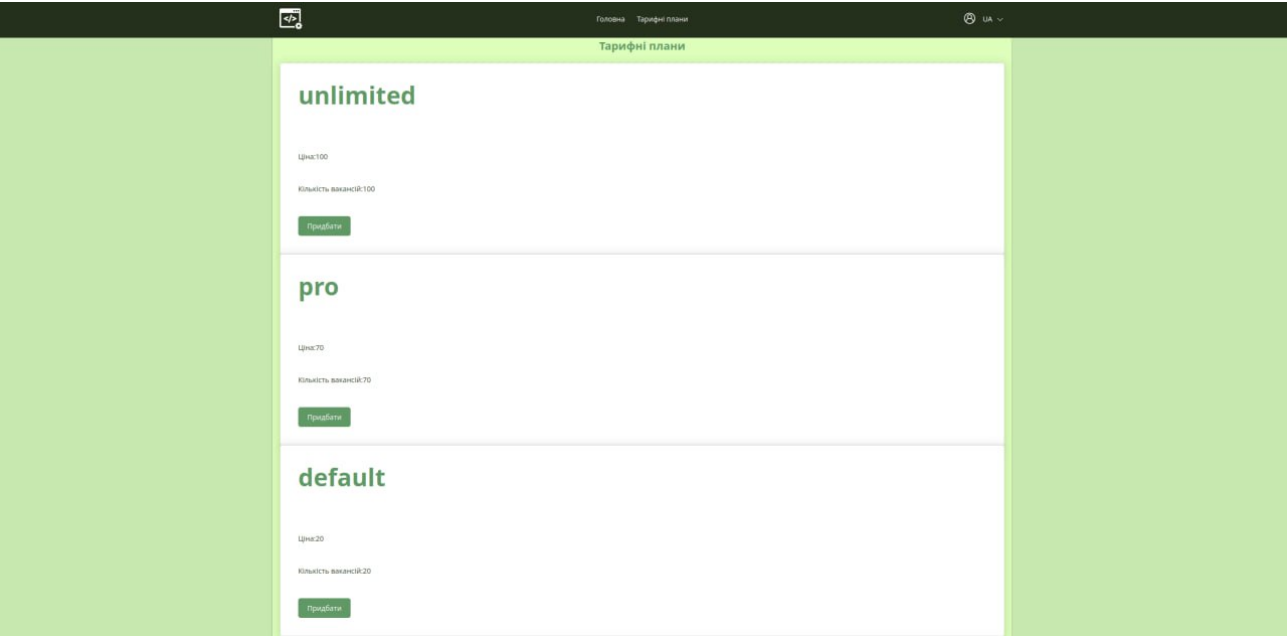


Рисунок 3.20 - Сторінка перегляду тарифних планів

На рисунку 3.21 зображено форму для вибору компанії перед оплатою.

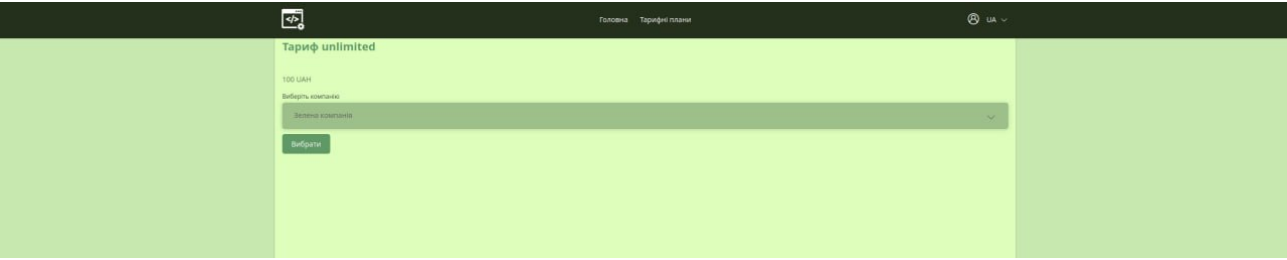


Рисунок 3.21 - Форма для вибору компанії для оплати

На рисунку 3.22 зображено сторінку оплати сервісу LiqPay.

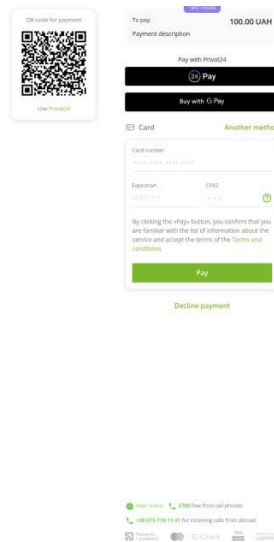


Рисунок 3.22 - Сторінка для оплати на сервісі LiqPay

На рисунку 3.23 зображено повідомлення про успішну оплату.

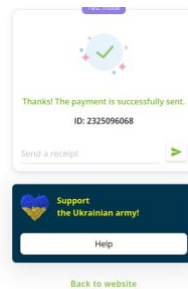


Рисунок 3.23 - Повідомлення про успішну оплату

Для адміністрації передбачено перегляд транзакцій та фільтрування для їх швидкого пошуку. На рисунку 3.24 зображено список транзакцій.

Головна

Тарифні плани

🇺🇦

UA

▼

Транзакції

Фільтр

12

Кількість: 5

Фільтрувати за полями:

Назва компанії

ID компанії

Назва тарифного плану

ID тарифного плану

Сума

Валюта

Статус транзакції

Дата проведення:

Дата від

Дата до

Очистити фільтр

ID	Компанія	Тарифний план	Сума	Статус	Дата
26a82153-6c7b-4215-ae3b-2472357e3360	Компанія тестування	pro	70.00UAH	Pending	12.06.2023 12:13
483634af-4c51-4b27-ae2a-77ee7284e96b	Зелена компанія	unlimited	100.00UAH	Pending	12.06.2023 12:12
5b9f9e83-ca3b-41ea-b54f-af5a8156955c	Зелена компанія	pro	70.00UAH	Pending	12.06.2023 12:13
79094435-c716-4801-b804-1d531d49264	Компанія тестування	unlimited	100.00UAH	Pending	10.06.2023 14:49
98727987-4baf-4445-b32f-508a10928ce1	Компанія тестування	unlimited	100.00UAH	Pending	12.06.2023 12:12

Рисунок 3.24 - Список транзакцій

Для того щоб адміністрація могла оперативно реагувати на створення нових компаній щоб їх верифікувати, адміністраторів додають в телеграм канал, в який будуть приходити сповіщення. На рисунку 3.25 зображено телеграм-канал для сповіщень.

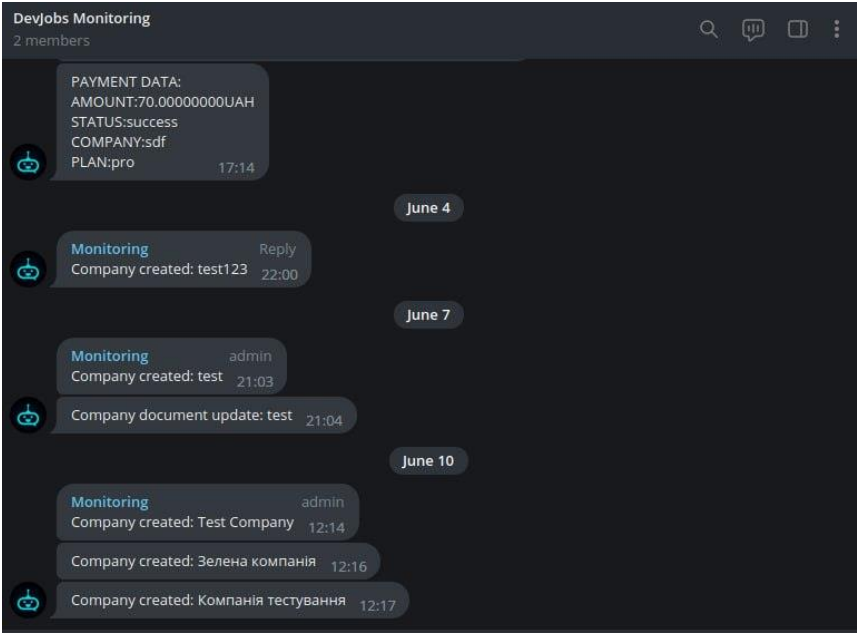


Рисунок 3.25 - Сервісні сповіщення в телеграм каналі

Також для адміністрації передбачено перегляд списку користувачів з можливістю їх блокувати та розблоковувати. На рисунку 3.26 зображено сторінку списку користувачів.

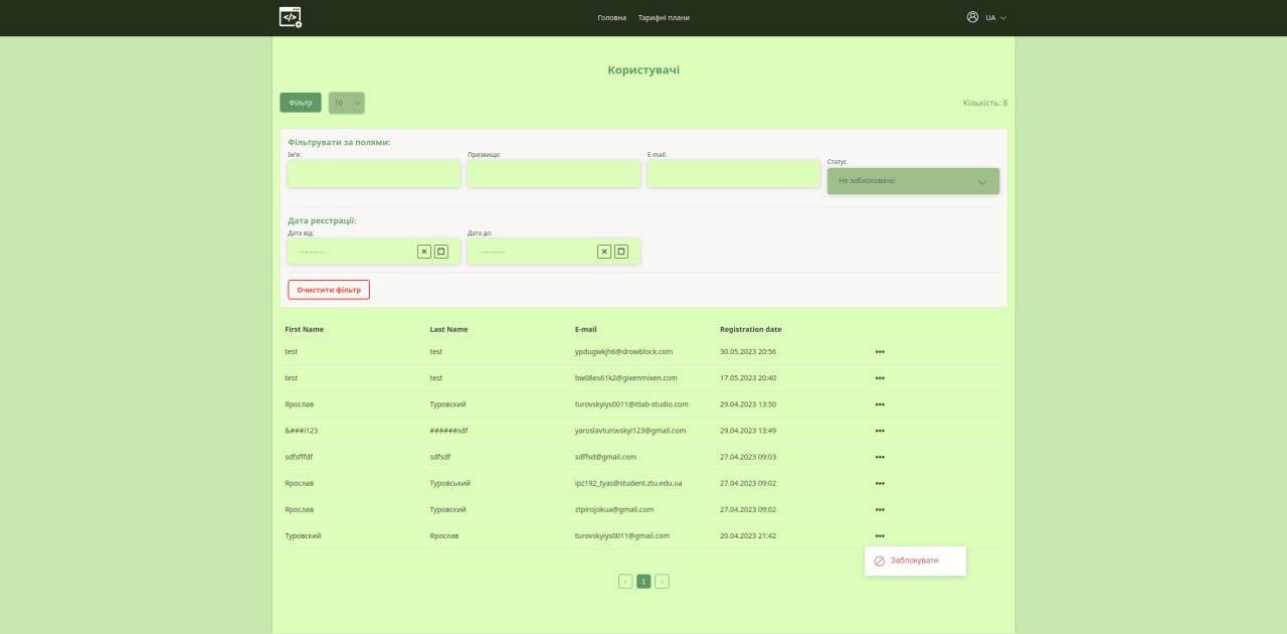


Рисунок 3.26 - Сторінка списку користувачів

3.3. Тестування роботи сайту

Якщо роботодавець спробує створити компанію з пустими полями, йому виведуться помилки під полями для введення обов'язкових полів. На рисунку 3.27 зображено валідація на формі створення компаній.

The screenshot shows a web form titled "Створення нової компанії" (Create new company). The form has several fields, each with a red border indicating a validation error. The fields are: "Назва:" (Name), "Логотип" (Logo), "Опис:" (Description), and "Документ:" (Document). The "Назва:" field has a red border and a red error message "Це поле є обов'язковим для заповнення" (This field is required for completion). The "Логотип" field has a red border and a red error message "Оберіть картинку" (Select a picture). The "Опис:" field has a red border and a red error message "Опис вашої компанії" (Description of your company). The "Документ:" field has a red border and a red error message "Це поле є обов'язковим для заповнення" (This field is required for completion). Below the "Документ:" field, there is a yellow box with a warning icon and text: "Для того щоб створити компанію, Вам потрібно завантажити один з переліку документів, для того щоб адмін мав змогу верифікувати легітимність Вашої компанії. Виберіть один з документів (Свідоцтво про державну реєстрацію, Податковий номер або інший документ, що підтвердить легітимність компанії)". At the bottom of the form, there are two buttons: "Завантажити" (Upload) and "Створити" (Create).

Рисунок 3.27 - Виведення помилок при створенні компанії з пустими полями

Якщо користувач спробує створити контактну інформацію компанії, не вказавши її тип або значення, він отримає повідомлення з підказкою. Також якщо користувач спробує створити більше 5 контактів для своєї компанії він отримає про це повідомлення. На рисунку 3.28 зображено виведення повідомлення підчас спроби створити контакт не вказавши його тип.

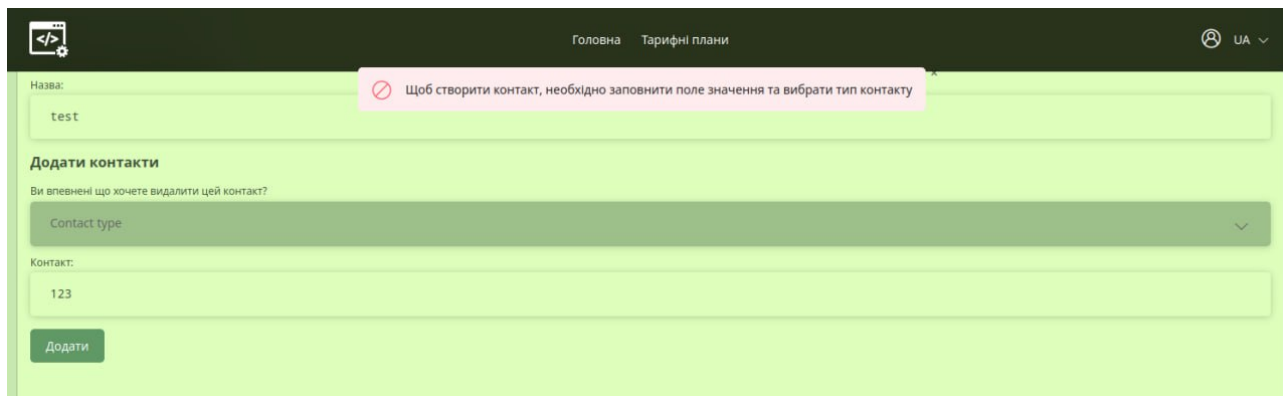


Рисунок 3.28 - Виведення повідомлення при спробі створити контакт не вказавши його тип

На рисунку 3.29 зображено виведення повідомлення про максимальну кількість контактів.

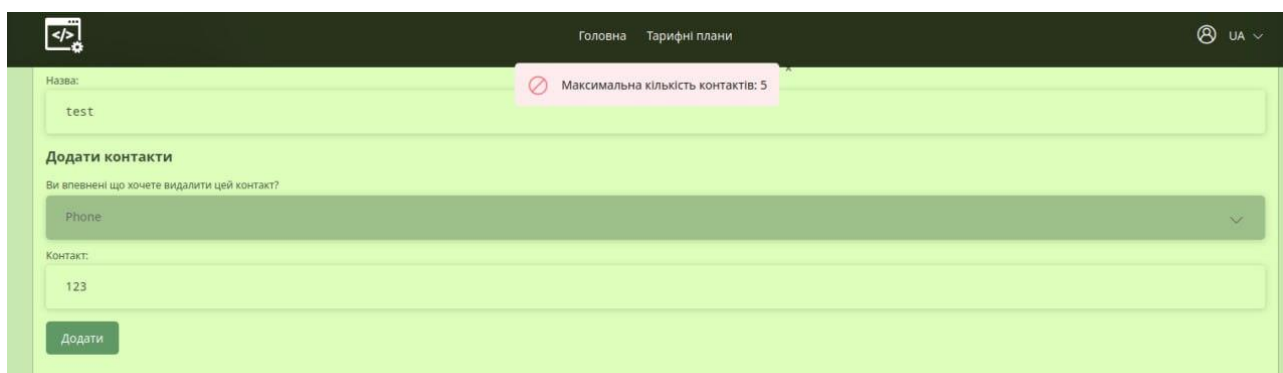


Рисунок 3.29 - Виведення повідомлення про максимальну кількість контактів для компанії

При створенні вакансії роботодавець має вказати всі обов'язкові поля та опис вакансії має бути не менше 20 символів. Також для створення вакансій було додано перевірку на наявність хоча б одної верифікованої компанії що має тарифний план.

На рисунку 3.30 зображено виведення помилок валідації підчас створення вакансії.

Рисунок 3.30 - Виведення помилок валідації підчас створення вакансії

На рисунку 3.31 зображено виведення помилки при спробі створити вакансії без наявності компаній.

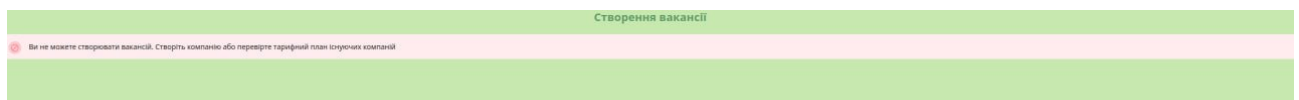


Рисунок 3.31 - Виведення помилки при спробі створити вакансії без наявності компаній

Підчас спілкування в діалогах додано валідацію на кількість символів для повідомлення. На рисунках 3.32 та 3.33 зображено виведення помилок валідації підчас написання повідомлень.

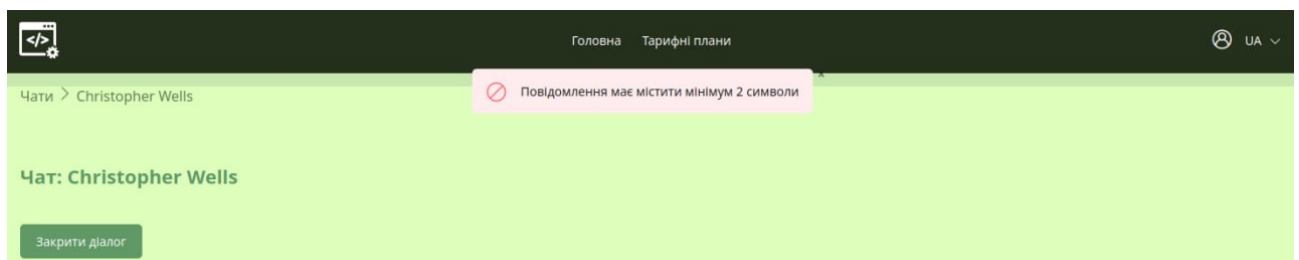


Рисунок 3.32 - Виведення помилок валідації підчас написання повідомлень

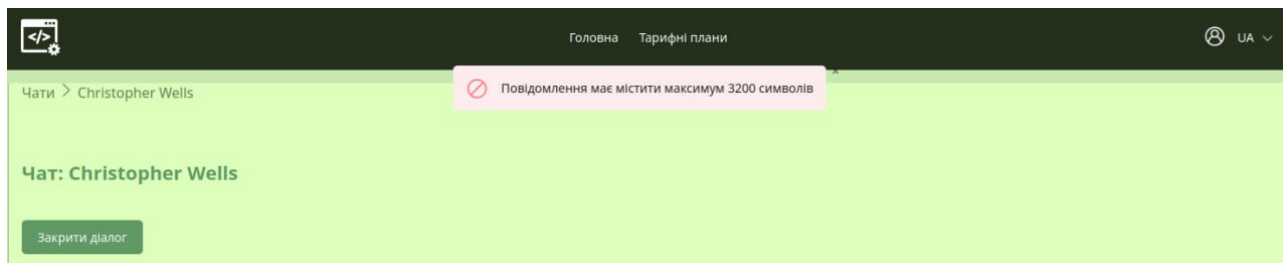


Рисунок 3.33 - Виведення помилок валідації під час написання повідомлень

Для створення резюме також було додано обробку помилок для полів. На завантаження файлу CV було додано перевірку чи є він формату pdf.

На рисунку 3.34 зображено виведення помилок валідації під час створення резюме.

Рисунок 3.34 - Виведення помилок валідації під час створення резюме

На рисунку 3.35 зображено виведення помилки під час завантаження файлу іншого розширення.

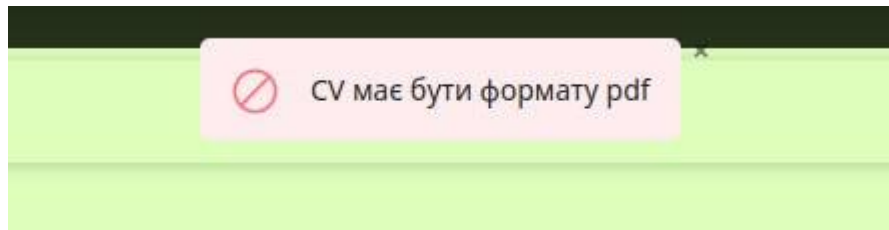


Рисунок 3.35 - Виведення помилки підчас завантаження файлу іншого розширення

Для користувачів, які ввімкнули двоетапну авторизацію, працює перевірка коду підчас авторизації. Для введення коду у користувача є три спроби. По вичерпанню трьох спроб його акаунт буде заблоковано. На рисунку 3.36 зображено виведення помилки при невдалих спробах введення коду.

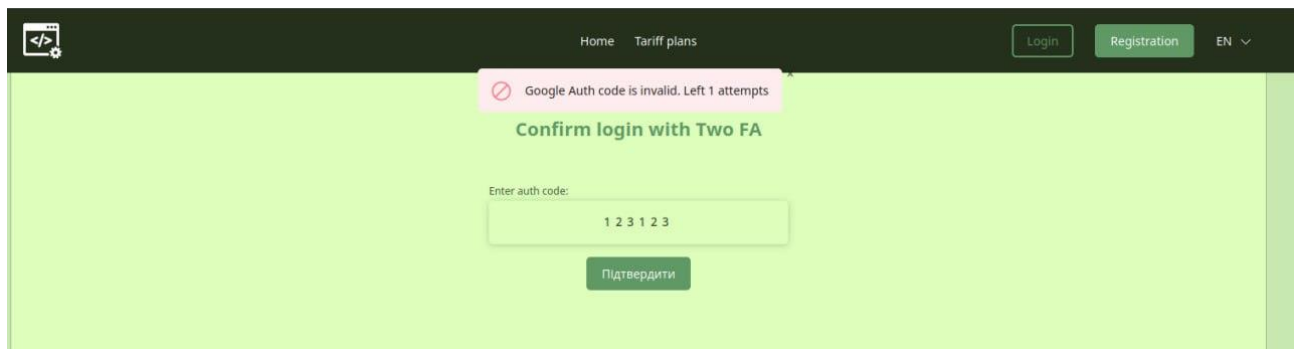


Рисунок 3.36 - Спроби введення неправильного коду

На рисунку 3.37 зображено виведення повідомлення про блокування акаунту.

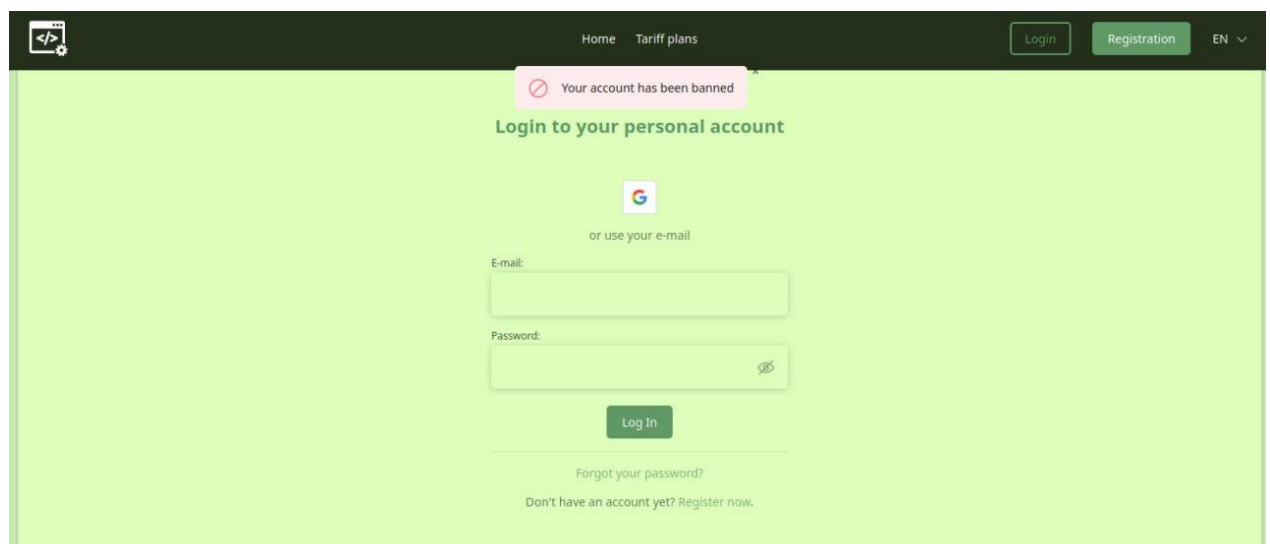


Рисунок 3.37 - Спроба входу в заблокований акаунт

Висновки до третього розділу

В даному розділі було детально описано процес встановлення та налаштування додатку, а також були визначені необхідні налаштування системи. Розкрито взаємодію між внутрішніми компонентами системи, а також представлено архітектуру програмного комплексу та його методи розгортання.

Додатково, викладено конкретні приклади та надано інструкції щодо роботи з розробленим сайтом. У цьому розділі основна увага приділена аналізу та опису розробленого програмного продукту.

ВИСНОВКИ

Підчас виконання випускної роботи бакалавра було спроектовано та реалізовано сайт з працевлаштування у сфері ІТ-технологій. Для успішної реалізації даної задачі було необхідно провести ретельний аналіз, а також розглянути різні методи та засоби, які можуть бути використані для її вирішення.

В першому розділі аналіз напрямків використання, дозволив виявити основні аспекти, які необхідно врахувати при розробці сайту. Підчас вивчення існуючих аналогів було виявлено що найважливішими характеристиками майбутньої системи є широкий вибір та зручність використання.

Після вибору оптимальної архітектури – клієнт-сервер з використанням RESTful API, були обрані підходящі технології для її реалізації, а саме Symfony, React та Docker. В якості аналогів було розглянуто такі технології як Laravel, Vue.js, Angular, Podman та LXD. Проте обрані технології є найбільш підходящими та сучасними для успішної реалізації поставлених завдань.

В другому розділі вебдодаток реалізує функціонал працевлаштування та пошуку фахівців в сфері ІТ-технологій. Для оптимального функціонування додатку, було розроблено детальну структуру бази даних MySQL з можливістю подальшого розширення та масштабування.

Детально вивчалися вимоги до системи, завдяки чому було розроблено алгоритми роботи додатку та візуалізовано їх в якості діаграм. В останньому пункті другого розділу було розглянуто аспекти реалізації важливих модулів системи.

В третьому розділі було детально описано процес встановлення, налаштування та розвертування додатку за допомогою Docker, а також були визначені необхідні налаштування системи. Розкрито взаємодію між внутрішніми компонентами системи, а також представлено архітектуру програмного комплексу та його методи розгортання.

Додатково, викладено конкретні приклади та надано інструкції щодо роботи з розробленим сайтом. У цьому розділі основна увага приділена аналізу та опису розробленого програмного продукту.

В результаті розробки було створено сайт з працевлаштування у сфері ІТ-технологій, який повністю задовольняє поставлені вимоги.

Завдяки використанню Docker, проект з легкістю можна розвернути на будь якій UNIX-подібній операційній системі. Цей проект може бути використаний як каркас для розробки більш масштабних проектів, які відрізнятимуться надійним рівнем безпеки, високою продуктивністю, зручністю використання та можливістю масштабування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. DOU [Електронний ресурс] – Режим доступу до ресурсу: <https://dou.ua/>.
2. Djinni [Електронний ресурс] – Режим доступу до ресурсу: <https://djinni.co/>.
3. Work.ua [Електронний ресурс] – Режим доступу до ресурсу: <https://www.work.ua/>.
4. RESTful API [Електронний ресурс] – Режим доступу до ресурсу: <https://aws.amazon.com/what-is/restful-api/>.
5. Документація Symfony [Електронний ресурс] – Режим доступу до ресурсу: <https://symfony.com/doc/current/index.html>.
6. Документація Doctrine [Електронний ресурс] – Режим доступу до ресурсу: <https://www.doctrine-project.org/projects/doctrine-orm/en/2.15/index.html>.
7. Документація MySQL [Електронний ресурс] – Режим доступу до ресурсу: <https://dev.mysql.com/doc/>.
8. Документація React [Електронний ресурс] – Режим доступу до ресурсу: <https://legacy.reactjs.org/docs/getting-started.html>.
9. ApiPlatform [Електронний ресурс] – Режим доступу до ресурсу: <https://api-platform.com/docs/core/>.
10. Mercure [Електронний ресурс] – Режим доступу до ресурсу: <https://mercure.rocks/>.
11. Документація Docker [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.docker.com/>.
12. NGINX [Електронний ресурс] – Режим доступу до ресурсу: <https://www.nginx.com/resources/glossary/nginx/>.
13. Документація Axios [Електронний ресурс] – Режим доступу до ресурсу: <https://axios-http.com/docs/intro>.

- 14.UML для бізнес-моделювання: для чого потрібні діаграми процесів
[Електронний ресурс] – Режим доступу до ресурсу:
<https://evergreens.com.ua/ua/articles/uml-diagrams.html>.
- 15.LiqPay [Електронний ресурс] – Режим доступу до ресурсу:
<https://www.liqpay.ua/en>.
- 16.Single Responsibility [Електронний ресурс] – Режим доступу до ресурсу:
<https://stackify.com/solid-design-principles/>.
- 17.Docker Compose [Електронний ресурс] – Режим доступу до ресурсу:
<https://docs.docker.com/compose/>.
- 18.JWT [Електронний ресурс] – Режим доступу до ресурсу: <https://jwt.io/>.

ДОДАТКИ

					ПЗ.КР.Б – 121 – 23 – ПЗ	
						62

Клас, що реалізує сутність Resume

```

<?php

namespace App\Entity;

use ApiPlatform\Core\Annotation\ApiFilter;
use ApiPlatform\Core\Annotation\ApiResource;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\BooleanFilter;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\RangeFilter;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;
use App\Action\CreateResumeAction;
use App\Action\EditResumeAction;
use App\Action\EditResumeMediaFilesAction;
use App\Action\GetAdditionalDataAction;
use App\Entity\Contacts\ResumeContacts;
use App\Entity\Traits\TimestampingTrait;
use App\Repository\ResumeRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
use Ramsey\Uuid\Uuid;
use Ramsey\Uuid\UuidInterface;
use Symfony\Component\Serializer\Annotation\Groups;
use Symfony\Component\Validator\Constraints as Assert;

#[ApiResource(
    collectionOperations: [
        'get' => [
            'security' => "is_granted('ROLE_ADMIN') or
is_granted('ROLE_WORKER') or is_granted('ROLE_EMPLOYER')",
            'method' => "GET",
            'normalization_context' => ['groups' => ['get:collection:resume']]
        ],
        'post' => [
            'method' => "POST",
            'denormalization_context' => ['groups' =>
['post:collection:resume']],
            'normalization_context' => ['groups' => ['get:item:resume']],
            'controller' => CreateResumeAction::class,
            'security' => "is_granted('ROLE_WORKER')"
        ],
        'editMediaFiles' => [
            'security' => "is_granted('ROLE_WORKER') or
is_granted('ROLE_ADMIN')",
            'method' => "POST",
            'path' => "/edit-media-files",
            'normalization_context' => ['groups' => ['get:item:resume']],
            'controller' => EditResumeMediaFilesAction::class
        ],
        'getAdditionalData' => [
            'method' => "GET",
            'path' => "/get-additional-data",
            'security' => "is_granted('ROLE_ADMIN') or
is_granted('ROLE_WORKER') or is_granted('ROLE_EMPLOYER')",
            'controller' => GetAdditionalDataAction::class,
            'normalization_context' => ['groups' => ['get:collection:additional-
data']]
        ],
    ],
    itemOperations: [

```

```

        'get'      => [
            'method'      => "GET",
            'normalization_context' => ['groups' => ['get:item:resume']],
            'security'     => "is_granted('ROLE_ADMIN') or
is_granted('ROLE_WORKER') or is_granted('ROLE_EMPLOYER')",
        ],
        'put'      => [
            'security'     => "(is_granted('ROLE_WORKER') and
object.getWorker() == user) or is_granted('ROLE_ADMIN')",
            'method'      => "PUT",
            'denormalization_context' => ['groups' =>
['put:collection:resume']],
            'normalization_context'  => ['groups' => ['get:item:resume']],
            'controller'    => EditResumeAction::class
        ],
        'delete' => [
            'method'      => "DELETE",
            'security'    => "is_granted('ROLE_ADMIN')"
        ]
    ]
})]
#[ApiFilter(SearchFilter::class, properties: [
    'firstname'      => 'partial',
    'lastname'       => 'partial',
    'worker.id'      => 'exact',
    'category'       => 'exact',
    'city'           => 'exact',
    'englishLevel'   => 'exact',
    'jobType'        => 'exact'
])]
#[ApiFilter(RangeFilter::class, properties: [
    'experience',
    'expectedSalary'
])]
#[ApiFilter(BooleanFilter::class, properties: [
    "isPublished"
])]
/**
 * @ORM\Entity(repositoryClass=ResumeRepository::class)
 * @ORM\HasLifecycleCallbacks
 */
class Resume
{
    use TimestampingTrait;

    /**
     * @var UuidInterface
     * @ORM\Id()
     * @ORM\Column(type="uuid")
     * @Groups({
     *     "get:item:resume",
     *     "get:collection:resume",
     *     "get:collection:vacancy-resume",
     *     "get:item:dialog"
     * })
     */
    private UuidInterface $id;

    /**
     * @Assert\NotBlank()
     * @ORM\Column(type="string", length=255)
     * @Groups({
     *     "get:item:resume",
     *     "get:collection:resume",

```

```

*      "post:collection:resume",
*      "put:collection:resume",
*      "get:collection:vacancy-resume",
*      "get:collection:dialog",
*      "get:item:dialog"
*  })
*  @Assert\Length(
*      min = 1,
*      max = 30,
*      minMessage = "Ім'я має бути мінімум {{ limit }} символів",
*      maxMessage = "Ім'я має бути максимум {{ limit }} символів"
*  )
*  @Assert\Regex(
*      pattern="/^[a-zA-Za-ягіїуєєщхыгьрэтьюА-ЯГІІУЄЄШХЫГЬРЭТЬЮ]{1,30}$/",
*      match=true,
*      message="Поле Ім'я має недопустимі символи"
*  )
*/
private string $firstname;

/**
*  @Assert\NotBlank()
*  @ORM\Column(type="string", length=255)
*  @Groups({
*      "get:item:resume",
*      "get:collection:resume",
*      "post:collection:resume",
*      "put:collection:resume",
*      "get:collection:vacancy-resume",
*      "get:collection:dialog",
*      "get:item:dialog"
*  })
*  @Assert\Length(
*      min = 1,
*      max = 30,
*      minMessage = "Прізвище має бути мінімум {{ limit }} символів",
*      maxMessage = "Прізвище має бути максимум {{ limit }} символів"
*  )
*  @Assert\Regex(
*      pattern="/^[a-zA-Za-ягіїуєєщхыгьрэтьюА-ЯГІІУЄЄШХЫГЬРЭТЬЮ]{1,30}$/",
*      match=true,
*      message="Поле Прізвище має недопустимі символи"
*  )
*/
private string $lastname;

/**
*  @Assert\NotBlank()
*  @ORM\Column(type="string", length=255)
*  @Groups({
*      "get:item:resume",
*      "get:collection:resume",
*      "post:collection:resume",
*      "put:collection:resume",
*      "get:collection:vacancy-resume",
*      "get:collection:dialog",
*      "get:item:dialog"
*  })
*  @Assert\Length(
*      min = 6,
*      max = 50,
*      minMessage = "Email має бути мінімум {{ limit }} символів",
*      maxMessage = "Email має бути максимум {{ limit }} символів"
*  )
*/

```

```

    * @Assert\Regex(
    *     pattern="/^[_a-z0-9-]+\.(\\.[_a-z0-9-])*[a-z-]+((\\.[a-z-])*(\\.[a-
z]{2,}){1,5})$/",
    *     match=true,
    *     message="Поле Email має недопустимі символи"
    * )
    */
private string $email;

/**
 * @var string
 * @ORM\Column(type="string", length=100)
 * @Assert\NotBlank()
 * @Groups({
 *     "post:collection:resume",
 *     "get:collection:resume",
 *     "get:item:resume",
 *     "put:collection:resume",
 *     "get:collection:vacancy-resume"
 * })
 */
private string $category;

/**
 * @Assert\NotBlank()
 * @Assert\Positive()
 * @ORM\Column(type="integer")
 * @Groups({
 *     "post:collection:resume",
 *     "get:collection:resume",
 *     "get:item:resume",
 *     "put:collection:resume",
 *     "get:collection:vacancy-resume"
 * })
 */
private int $expectedSalary;

/**
 * @ORM\Column(type="float", nullable=true)
 * @Groups({
 *     "post:collection:resume",
 *     "get:collection:resume",
 *     "get:item:resume",
 *     "put:collection:resume",
 *     "get:collection:vacancy-resume"
 * })
 */
private float $experience;

/**
 * @var null|string
 * @ORM\Column(type="string", length=100, nullable=true)
 * @Groups({
 *     "post:collection:resume",
 *     "get:collection:resume",
 *     "get:item:resume",
 *     "put:collection:resume",
 *     "get:collection:vacancy-resume"
 * })
 */
private ?string $city;

/**
 * @var null|string

```

```

    * @ORM\Column(type="string", length=100, nullable=true)
    * @Groups({
    *     "post:collection:resume",
    *     "get:collection:resume",
    *     "get:item:resume",
    *     "put:collection:resume",
    *     "get:collection:vacancy-resume"
    * })
    */
private ?string $englishLevel;

/**
 * @var null|string
 * @ORM\Column(type="string", length=100, nullable=true)
 * @Groups({
 *     "post:collection:resume",
 *     "get:collection:resume",
 *     "get:item:resume",
 *     "put:collection:resume",
 *     "get:collection:vacancy-resume"
 * })
 */
private ?string $jobType;

/**
 * @var null|string
 * @ORM\Column(type="string", length=100, nullable=true)
 * @Groups({
 *     "post:collection:resume",
 *     "get:collection:resume",
 *     "get:item:resume",
 *     "put:collection:resume"
 * })
 */
private ?string $portfolio;

/**
 * @Assert\NotBlank()
 * @ORM\Column(type="string", length=3200)
 * @Groups({
 *     "post:collection:resume",
 *     "get:collection:resume",
 *     "get:item:resume",
 *     "put:collection:resume"
 * })
 * @Assert\Length(
 *     min = 3,
 *     max = 3200,
 *     minMessage = "Опис досвіду має бути мінімум {{ limit }} символів",
 *     maxMessage = "Опис досвіду має бути максимум {{ limit }} символів"
 * )
 */
private string $experienceDescription;

/**
 * @Assert\NotBlank()
 * @ORM\Column(type="string", length=3200)
 * @Groups({
 *     "post:collection:resume",
 *     "get:collection:resume",
 *     "get:item:resume",
 *     "put:collection:resume"
 * })
 * @Assert\Length(

```

```

        *         min = 3,
        *         max = 3200,
        *         minMessage = "Опис очікувань має бути мінімум {{ limit }} символів",
        *         maxMessage = "Опис очікувань має бути максимум {{ limit }} символів"
        * )
        */
private string $expectationDescription;

/**
 * @Assert\NotBlank()
 * @ORM\Column(type="string", length=3200)
 * @Groups({
 *     "post:collection:resume",
 *     "get:collection:resume",
 *     "get:item:resume",
 *     "put:collection:resume"
 * })
 * @Assert\Length(
 *     min = 3,
 *     max = 3200,
 *     minMessage = "Опис досягнень має бути мінімум {{ limit }} символів",
 *     maxMessage = "Опис досягнень має бути максимум {{ limit }} символів"
 * )
 */
private string $achievementsDescription;

/**
 * @ORM\OneToMany(targetEntity=ResumeContacts::class, mappedBy="resume",
cascade={"remove"})
 * @ORM\JoinColumn()
 * @Groups({
 *     "get:item:resume",
 *     "get:collection:resume",
 * })
 */
private Collection $resumeContacts;

/**
 * @var MediaObject|null
 * @ORM\OneToOne(targetEntity=MediaObject::class, mappedBy="resumePhoto",
orphanRemoval=true, cascade={"remove"})
 * @Groups({
 *     "get:collection:resume",
 *     "get:item:resume"
 * })
 */
private ?MediaObject $photo;

/**
 * @var MediaObject|null
 * @ORM\OneToOne(targetEntity=MediaObject::class, mappedBy="resumeCv",
orphanRemoval=true, cascade={"remove"})
 * @Groups({
 *     "get:collection:resume",
 *     "get:item:resume"
 * })
 */
private ?MediaObject $cv;

/**
 * @var bool
 * @ORM\Column(type="boolean")
 * @Groups({
 *     "post:collection:resume",

```

```

        *      "get:collection:resume",
        *      "get:item:resume",
        *      "put:collection:resume"
        *  })
        */
private bool $isPublished;

/**
 * @ORM\OneToMany(targetEntity=ResumeTechnologies::class, mappedBy="resume",
cascade={"remove"})
 * @ORM\JoinColumn()
 * @Groups({
 *      "get:collection:resume",
 *      "get:item:resume"
 *  })
 */
private Collection $technologies;

/**
 * @var User
 * @ORM\ManyToOne(targetEntity=User::class, inversedBy="resume")
 * @Groups({
 *      "get:collection:resume",
 *      "get:item:resume",
 *      "get:collection:vacancy-resume"
 *  })
 */
private User $worker;

/**
 * @var Collection
 * @ORM\OneToMany(targetEntity=VacancyResume::class, mappedBy="resume",
orphanRemoval=true, cascade={"remove"})
 */
private Collection $vacancies;

/**
 * @ORM\OneToMany(targetEntity=Dialog::class, mappedBy="worker",
cascade={"remove"})
 * @ORM\JoinColumn()
 */
private Collection $dialogs;

/**
 * Company constructor.
 */
public function __construct()
{
    $this->id = Uuid::uuid4();
    $this->technologies = new ArrayCollection();
    $this->resumeContacts = new ArrayCollection();
    $this->vacancies = new ArrayCollection();
    $this->dialogs = new ArrayCollection();
    $this->isPublished = false;
}

/**
 * @return UuidInterface|null
 */
public function getId(): ?UuidInterface
{
    return $this->id;
}

```

```

/**
 * @return Collection
 */
public function getResumeContacts(): Collection
{
    return $this->resumeContacts;
}

/**
 * @param ResumeContacts $resumeContacts
 * @return $this
 */
public function addResumeContacts(ResumeContacts $resumeContacts): self
{
    if (!$this->resumeContacts->contains($resumeContacts)) {
        $this->resumeContacts[] = $resumeContacts;
        $resumeContacts->setResume($this);
    }

    return $this;
}

/**
 * @return string
 */
public function getFirstname(): string
{
    return $this->firstname;
}

/**
 * @param string $firstname
 * @return Resume
 */
public function setFirstname(string $firstname): self
{
    $this->firstname = $firstname;

    return $this;
}

/**
 * @return string
 */
public function getLastname(): string
{
    return $this->lastname;
}

/**
 * @param string $lastname
 * @return Resume
 */
public function setLastname(string $lastname): self
{
    $this->lastname = $lastname;

    return $this;
}

/**
 * @return string
 */
public function getCategory(): string

```

```

    {
        return $this->category;
    }

    /**
     * @param string $category
     * @return Resume
     */
    public function setCategory(string $category): self
    {
        $this->category = $category;

        return $this;
    }

    /**
     * @return int
     */
    public function getExpectedSalary(): int
    {
        return $this->expectedSalary;
    }

    /**
     * @param int $expectedSalary
     * @return Resume
     */
    public function setExpectedSalary(int $expectedSalary): self
    {
        $this->expectedSalary = $expectedSalary;

        return $this;
    }

    /**
     * @return float
     */
    public function getExperience(): float
    {
        return $this->experience;
    }

    /**
     * @param float $experience
     * @return Resume
     */
    public function setExperience(float $experience): self
    {
        $this->experience = $experience;

        return $this;
    }

    /**
     * @return string|null
     */
    public function getCity(): ?string
    {
        return $this->city;
    }

    /**
     * @param string|null $city
     * @return Resume

```

```

        */
public function setCity(?string $city): self
{
    $this->city = $city;

    return $this;
}

/**
 * @return string|null
 */
public function getEnglishLevel(): ?string
{
    return $this->englishLevel;
}

/**
 * @param string|null $englishLevel
 * @return Resume
 */
public function setEnglishLevel(?string $englishLevel): self
{
    $this->englishLevel = $englishLevel;

    return $this;
}

/**
 * @return string|null
 */
public function getJobType(): ?string
{
    return $this->jobType;
}

/**
 * @param string|null $jobType
 * @return Resume
 */
public function setJobType(?string $jobType): self
{
    $this->jobType = $jobType;

    return $this;
}

/**
 * @return string
 */
public function getExperienceDescription(): string
{
    return $this->experienceDescription;
}

/**
 * @param string $experienceDescription
 * @return Resume
 */
public function setExperienceDescription(string $experienceDescription):
self
{
    $this->experienceDescription = $experienceDescription;

    return $this;
}

```

```

    }

    /**
     * @return string
     */
    public function getExpectationDescription(): string
    {
        return $this->expectationDescription;
    }

    /**
     * @param string $expectationDescription
     * @return Resume
     */
    public function setExpectationDescription(string $expectationDescription):
self
    {
        $this->expectationDescription = $expectationDescription;

        return $this;
    }

    /**
     * @return string
     */
    public function getAchievementsDescription(): string
    {
        return $this->achievementsDescription;
    }

    /**
     * @param string $achievementsDescription
     * @return Resume
     */
    public function setAchievementsDescription(string $achievementsDescription):
self
    {
        $this->achievementsDescription = $achievementsDescription;

        return $this;
    }

    /**
     * @return MediaObject|null
     */
    public function getPhoto(): ?MediaObject
    {
        return $this->photo;
    }

    /**
     * @param MediaObject|null $photo
     * @return Resume
     */
    public function setPhoto(?MediaObject $photo): self
    {
        $this->photo = $photo;

        $photo?->setResumePhoto($this);

        return $this;
    }

    /**

```

```

    * @return MediaObject|null
    */
    public function getCv(): ?MediaObject
    {
        return $this->cv;
    }

    /**
     * @param MediaObject|null $cv
     * @return Resume
     */
    public function setCv(?MediaObject $cv): self
    {
        $this->cv = $cv;

        $cv?->setResumeCv($this);

        return $this;
    }

    /**
     * @return bool
     */
    public function getIsPublished(): bool
    {
        return $this->isPublished;
    }

    /**
     * @param bool $isPublished
     * @return Resume
     */
    public function setIsPublished(bool $isPublished): self
    {
        $this->isPublished = $isPublished;

        return $this;
    }

    /**
     * @return Collection
     */
    public function getTechnologies(): Collection
    {
        return $this->technologies;
    }

    /**
     * @param ResumeTechnologies $technologies
     * @return $this
     */
    public function addTechnologies(ResumeTechnologies $technologies): self
    {
        if (!$this->technologies->contains($technologies)) {
            $this->technologies[] = $technologies;
            $technologies->setResume($this);
        }

        return $this;
    }

    /**
     * @return User
     */

```

```

public function getWorker(): User
{
    return $this->worker;
}

/**
 * @param User $worker
 * @return Resume
 */
public function setWorker(User $worker): self
{
    $this->worker = $worker;

    $worker->setResume($this);

    return $this;
}

/**
 * @return string
 */
public function getEmail(): string
{
    return $this->email;
}

/**
 * @param string $email
 * @return Resume
 */
public function setEmail(string $email): self
{
    $this->email = $email;

    return $this;
}

/**
 * @return string|null
 */
public function getPortfolio(): ?string
{
    return $this->portfolio;
}

/**
 * @param string|null $portfolio
 * @return Resume
 */
public function setPortfolio(?string $portfolio): self
{
    $this->portfolio = $portfolio;

    return $this;
}

/**
 * @return Collection
 */
public function getVacancies(): Collection
{
    return $this->vacancies;
}

```

```

    /**
     * @return Collection
     */
    public function getDialogs(): Collection
    {
        return $this->dialogs;
    }
}

```

Клас, що реалізує сутність Vacancy

<?php

namespace App\Entity;

use ApiPlatform\Core\Annotation\ApiFilter;

use ApiPlatform\Core\Annotation\ApiResource;

use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;

use App\Action\ApplyToVacancyAction;

use App\Action\CreateVacancyAction;

use App\Action\GetResumeAdditionalDataAction;

use App\Entity\Traits\TimestampingTrait;

use App\Repository\VacancyRepository;

use Doctrine\Common\Collections\ArrayCollection;

use Doctrine\Common\Collections\Collection;

use Doctrine\ORM\Mapping as ORM;

use Ramsey\Uuid\Uuid;

use Ramsey\Uuid\UuidInterface;

use Symfony\Component\Serializer\Annotation\Groups;

use Symfony\Component\Validator\Constraints as Assert;

#[ApiResource(

collectionOperations: [

'get' => [

'security' => "is_granted('ROLE_ADMIN') or

is_granted('ROLE_WORKER') or is_granted('ROLE_EMPLOYER')",

'method' => "GET",

'normalization_context' => ['groups' =>

['get:collection:vacancies']]

],

'post' => [

'method' => "POST",

'denormalization_context' => ['groups' =>

['post:collection:vacancy']],

'normalization_context' => ['groups' => ['get:item:vacancy']],

'controller' => CreateVacancyAction::class,

'security' => "is_granted('ROLE_EMPLOYER')"

],

'getVacancyAdditionalData' => [

'method' => "GET",

'path' => "/get-resume-additional-data",

'security' => "is_granted('ROLE_ADMIN') or

is_granted('ROLE_WORKER') or is_granted('ROLE_EMPLOYER')",

'controller' => GetResumeAdditionalDataAction::class,

'normalization_context' => ['groups' => ['get:collection:additional-

data']]

],

],

itemOperations: [

'get' => [

'method' => "GET",

'normalization_context' => ['groups' => ['get:item:vacancy']],

'security' => "is_granted('ROLE_ADMIN') or

is_granted('ROLE_WORKER') or is_granted('ROLE_EMPLOYER')",

],

'put' => [

'security' => "(is_granted('ROLE_EMPLOYER') and

object.getCompany().getEmployer() == user) or is_granted('ROLE_ADMIN')",

'method' => "PUT",

```

        'denormalization_context' => ['groups' =>
['put:collection:vacancy']],
        'normalization_context'    => ['groups' => ['get:item:vacancy']],
        'controller'               => CreateVacancyAction::class
    ],
    'apply' => [
        'method'                   => "PUT",
        'normalization_context'    => ['groups' => ['get:item:vacancy']],
        'denormalization_context' => ['groups' => ['put:apply:vacancy']],
        'path'                    => "/apply-to-vacancy/{id}",
        'controller'              => ApplyToVacancyAction::class,
        'security'                => "is_granted('ROLE_WORKER')"
    ],
    'delete' => [
        'method'                   => "DELETE",
        'security' => "is_granted('ROLE_ADMIN') or
(is_granted('ROLE_EMPLOYER') and object.getCompany().getEmployer() == user)"
    ]
]
)]
#[ApiFilter(SearchFilter::class, properties: [
    'description'    => 'partial',
    'name'           => 'partial',
    'category'       => 'exact',
    'city'           => 'exact',
    'company.employer.id' => 'exact',
    'company.name'    => 'exact'
])]
/**
 * @ORM\Entity(repositoryClass=VacancyRepository::class)
 * @ORM\HasLifecycleCallbacks
 */
class Vacancy
{
    use TimestampingTrait;

    /**
     * @var UuidInterface
     * @ORM\Id()
     * @ORM\Column(type="uuid")
     * @Groups({
     *     "get:item:resume",
     *     "get:item:vacancy",
     *     "get:collection:vacancies"
     * })
     */
    private UuidInterface $id;

    /**
     * @Assert\NotBlank()
     * @ORM\Column(type="string", length=255)
     * @Groups({
     *     "post:collection:vacancy",
     *     "get:item:vacancy",
     *     "get:collection:vacancies",
     *     "put:collection:vacancy",
     *     "get:collection:vacancy-resume"
     * })
     * @Assert\Length(
     *     min = 3,
     *     max = 250,
     *     minMessage = "Назва має бути мінімум {{ limit }} символів",
     *     maxMessage = "Назва має бути максимум {{ limit }} символів"
     * )

```

```

    */
    private string $name;

    /**
     * @var string
     * @ORM\Column(type="string", length=100)
     * @Assert\Length(
     *     min = 3,
     *     max = 100,
     *     minMessage = "Категорія має бути мінімум {{ limit }} символів",
     *     maxMessage = "Категорія має бути максимум {{ limit }} символів"
     * )
     * @Assert\NotBlank()
     * @Groups({
     *     "post:collection:vacancy",
     *     "get:item:vacancy",
     *     "get:collection:vacancies",
     *     "put:collection:vacancy"
     * })
     */
    private string $category;

    /**
     * @var null|string
     * @ORM\Column(type="string", length=100, nullable=true)
     * @Groups({
     *     "post:collection:vacancy",
     *     "get:item:vacancy",
     *     "get:collection:vacancies",
     *     "put:collection:vacancy"
     * })
     */
    private ?string $city;

    /**
     * @Assert\NotBlank()
     * @ORM\Column(type="text")
     * @Groups({
     *     "post:collection:vacancy",
     *     "get:item:vacancy",
     *     "get:collection:vacancies",
     *     "put:collection:vacancy"
     * })
     * @Assert\Length(
     *     min = 3,
     *     max = 5000,
     *     minMessage = "Опис досвіду має бути мінімум {{ limit }} символів",
     *     maxMessage = "Опис досвіду має бути максимум {{ limit }} символів"
     * )
     */
    private string $description;

    /**
     * @ORM\ManyToOne(targetEntity=Company::class, inversedBy="vacancies")
     * @ORM\JoinColumn(nullable=false)
     * @Groups({
     *     "get:item:vacancy",
     *     "get:collection:vacancies"
     * })
     */
    private Company $company;

    /**
     * @var Collection

```

```

        * @ORM\OneToMany(targetEntity=VacancyResume::class, mappedBy="vacancy",
orphanRemoval=true, cascade={"remove"})
        */
        private Collection $applicants;

        /**
         * Company constructor.
         */
        public function __construct()
        {
            $this->id = Uuid::uuid4();
            $this->applicants = new ArrayCollection();
        }

        /**
         * @return UuidInterface|null
         */
        public function getId(): ?UuidInterface
        {
            return $this->id;
        }

        /**
         * @return string
         */
        public function getName(): string
        {
            return $this->name;
        }

        /**
         * @param string $name
         * @return Vacancy
         */
        public function setName(string $name): self
        {
            $this->name = $name;

            return $this;
        }

        /**
         * @return string
         */
        public function getCategory(): string
        {
            return $this->category;
        }

        /**
         * @param string $category
         * @return Vacancy
         */
        public function setCategory(string $category): self
        {
            $this->category = $category;

            return $this;
        }

        /**
         * @return string|null
         */
        public function getCity(): ?string

```

```

    {
        return $this->city;
    }

    /**
     * @param string|null $city
     * @return Vacancy
     */
    public function setCity(?string $city): self
    {
        $this->city = $city;

        return $this;
    }

    /**
     * @return string
     */
    public function getDescription(): string
    {
        return $this->description;
    }

    /**
     * @param string $description
     * @return Vacancy
     */
    public function setDescription(string $description): self
    {
        $this->description = $description;

        return $this;
    }

    /**
     * @return Company
     */
    public function getCompany(): Company
    {
        return $this->company;
    }

    /**
     * @param Company $company
     * @return Vacancy
     */
    public function setCompany(Company $company): self
    {
        $this->company = $company;

        return $this;
    }

    /**
     * @return Collection
     */
    public function getApplicants(): Collection
    {
        return $this->applicants;
    }
}

```