КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра математичної інформатики

Кваліфікаційна робота на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

Веб-додаток для автоматизації процесів найму, відбору та централізації інформації про працівників з використанням машинного навчання та хмарних технологій

Виконав студент 4-го курсу

Микола ЛЕВАДНИЙ

Науковий керівник:
професор, кандидат фіз.-мат. наук

Ірина ВЕРГУНОВА

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

Студент

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики
«____» _____

2022 р., протокол № ___
Завідувач кафедри
Василь ТЕРЕЩЕНКО _____

Київ – 2022

РЕФЕРАТ

Обсяг роботи 62 - сторінок, 62 - ілюстрацій, 20 - джерел посилання.

Ключові слова: ВЕБ СЕРВІСИ, WEB API, RESTFUL SERVICE, .NET 5.0, ASP.NET CORE, КЛІЄНТ-СЕРВІС, ONION ARCHITECTURE, ANGULAR, ELASTICSEARCH, DOCKER, MS SQL, MONGODB, CQRS, AWS, AMAZON S3, AMAZON COMPREHEND, AMAZON TEXTRACT, JWT, SMTP.

Об’єкт дослідження*:* системи для автоматизації процесів найму, відбору та централізації інформації про працівників.

Мета роботи: створення системи (клієнт-сервіс) для відбору та найму працівників, з централізацією інформації для автоматизування процесів з використанням машинного навчання та хмарних технологій з найкращим функціоналом для додавання працівників.

Методи та інструменти дослідження*:* аналіз існуючих систем, аналіз існуючих принципів архітектури проєкту, існуючих шаблонів програмування, ключових положень та принципів щодо написання коду, аналіз способів оптимізації запитів, аналіз material дизайну [1], аналіз способів захисту авторизації, аналіз існуючих моделей для вирішення задачі, аналіз існуючих баз даних для зберігання файлів. Інструменти розроблення: інтегроване середовище розробки Microsoft Visual Studio Community 2019 16.11.8, мова програмування C# 11.0 на базі платформи .NET 5.0, Docker Desktop 4.1.1, Visual Studio Code 1.67.2, Aws console, MS SQL SERVER 11.4.7462.6, MongoDB Compass 5.0.3, GitHub Desktop 3.0.0.

Результати роботи*:* вивчено сучасні методи побудови веб-додатків; написано веб представлення та веб сервіс; розроблена та реалізована onion архітектура проєкту; створено модель для розпізнавання інформації про працівника; створено ряд шляхів для автоматизації відбору працівників; розроблена та реалізована The Command and Query Responsibility Segregation парадигма; вивчені та використані сучасні підходи до роботи з базами даних.

ПЕРЕЛІК СКОРОЧЕНЬ

MS SQL - Microsoft SQL Server

CQRS - The Command and Query Responsibility Segregation

JWT - JSON Web Token

SMTP - Simple Mail Transfer Protocol

AWS - Amazon Web Services

ATS – Applicant Tracking System

HR – рекрутер

ORM - Object-Relational Mapping

DI – Dependency injection

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта розробки. Наразі існує безліч ATS сервісів (Manatal, HURMA, Recrutiee). Основною функцією ATS є забезпечення центрального розташування інформації та бази даних для зменшення зусиль компанії з підбору персоналу. ATS призначений для кращої допомоги в управлінні резюме та інформацією про кандидатів. Дані збираються або з внутрішніх заяв через інтерфейс ATS на веб-сайті компанії, або від кандидатів. Більшість дошок вакансій і резюме (Reed Online, LinkedIn.com, Monster.com, Hotjobs, CareerBuilder, Indeed.com) співпрацюють з постачальниками програмного забезпечення ATS для підтримки аналізу та легкої міграції даних з однієї системи в іншу. Новіші системи відстеження заявників є платформою як послугою, більшість з яких мають точки інтеграції, які дозволяють постачальникам інших технологій рекрутингу легко підключатися.

ATS системи користуються популярністю і з кожним роком на ринці з’являються нові продукти, з новими підходам до автоматизації та додавання кандидатів до системи. З розвиненням подібних систем також розвиваються методи автоматизації. Так наразі для спрощення HR процесів на різних платформах використовуються такі підходи як: машинне навчання, автоматизація надсилання повідомлень на пошту, додатки до браузерів для діставання інформації про кандидатів з сайтів-резюме, парсинг файлів с даними.

Також для підтримки великих проектів створюються декілька проектів та додаткові модулі для функціонування системи. Так для графічного відображення системи створюють frontend-додаток який використовує технології для спрощеної побудови користувацького додатку. В свою чергу для підтримки frontend-додатку створюється веб-сервіс який приймає та опрацьовує звернення. Також можливе створення окремих модулів, як додаток до браузера, десктоп додаток і т.д.

Важливим є вибір архітектури. Архітектура повинна будуватись щоб найкраще відповідати вимогам до системи що створюється і зберігала при цьому здатність розширюватись та підтримуватись. Для цього використовують багаторівневі архітектури (для сервісів), та компонентно-сервісні (для frontend-додатків).

Актуальність теми. Сучасний світ розвивається дуже стрімкими темпами. Усі задачі автоматизуються і для них створюється зручний інтерфейс. Розробка сервісів - дуже актуальна річ у наші часи. Багато додатків так чи інакше використовують сервіси для отримання інформації. Задачі після автоматизації займають набагато менше часу, аніж коли цим займається людина. Наприклад, на сьогодні майже всі державні структури мають власні сервіси для отримання допомоги та запису на різного види послуги. Це набагато полегшило життя населення. Очевидно, що для пошуку працівників використовують сервіси для зручного управління, автоматизації та централізації інформації, без цього записування інформації займало б занадто багато часу у персоналу, що дуже не вигідно. Також багато провідних аутсорс компаній розробляють сервіси, тому це дуже гарна сфера для розвитку. При розробці сервісів використовуються різні технології, в тому числі особливо розповсюдженим є набір технологій, які будемо використовувати в рішенні нашого завдання.

Мета й завдання роботи. Метою роботи була розробка повноцінного та підтримуваного веб-додатку для автоматизації процесів найму, відбору та централізації інформації про працівників з використанням машинного навчання та хмарних технологій.

Для досягнення поставленої мети вирішували такі завдання:

- обрати архітектуру проєкту, виділити його основні підчастини та поділити кожен на свої підмодулі;
- реалізувати основні сутності та сервіси для роботи з ними (бізнес-логіка);
- реалізувати RESTful Web API сервіс;

- Реалізувати веб-додаток додаток для зручної роботи із сервісом;
- Реалізація авторизації зі збереженням сессії;
- Натренувати модель для діставання інформації з резюме;
- Розробка google chrome розширення для отримання інформації з сервісу Linkedin.

Об’єкт і методи дослідження або розроблення.

Об’єктом наших досліджень була архітектурна розробка повноцінного проєкту для автоматизації процесів найму, відбору та централізації інформації про працівників.

Методами дослідження були: аналіз існуючих сервісів, аналіз предметної області, розробка архітектури з можливістю розширення та підтримування.

Можливі сфери застосування. Проєкт реалізує зручний та зрозумілий інтерфейс для управління процесами найму та підтримує процес найму за сучасними вимогами, тому він може використовуватися у роботі як на приватних фірмах, так і на державних підприємствах. Планується практичне використання системи у відкритому доступі.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ РОЗВ'ЯЗАННЯ ЗАДАЧІ

1.1 BackEnd

Існує безліч різних видів і типів архітектур, які успішно застосовуються. Однією з найбільш популярних і в той же час застосовуваних для великих проєктів є ONION Architecture описана Р. Мартином в книжці «Чиста архітектура» [2]. Саме цю архітектуру я використав при реалізації проекту.

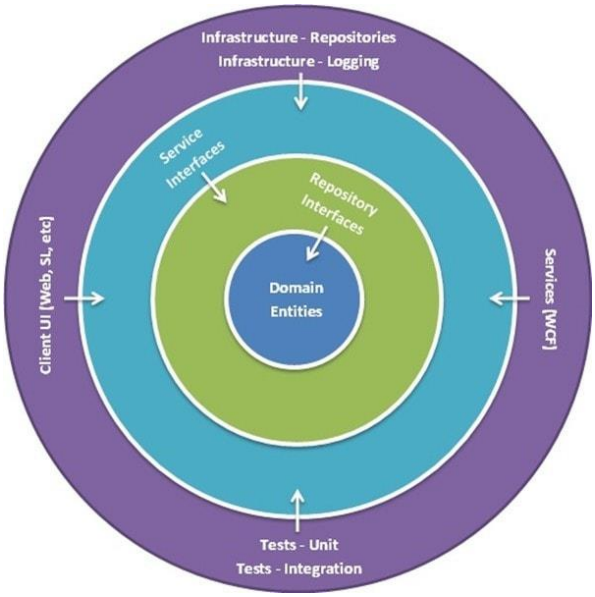


Рисунок 1.1 Onion архітектура

Ця архітектура забезпечує надійність та зручність у модифікації (Рис. 1.1).

Для розробки програмних рівнів додатку нам потрібна можливість розробки веб-сервісів, пакети для роботи з різними типами баз даних, відправкою листів, використання хмарних технологій.

Одна з найпоширеніших мов програмування, яка може забезпечити весь потрібній нам функціонал - це C# стандарту 11.0 та платформа .NET 5.0, яка надає можливість створення додатків фактично для будь-яких цілей (Рис. 1.2).

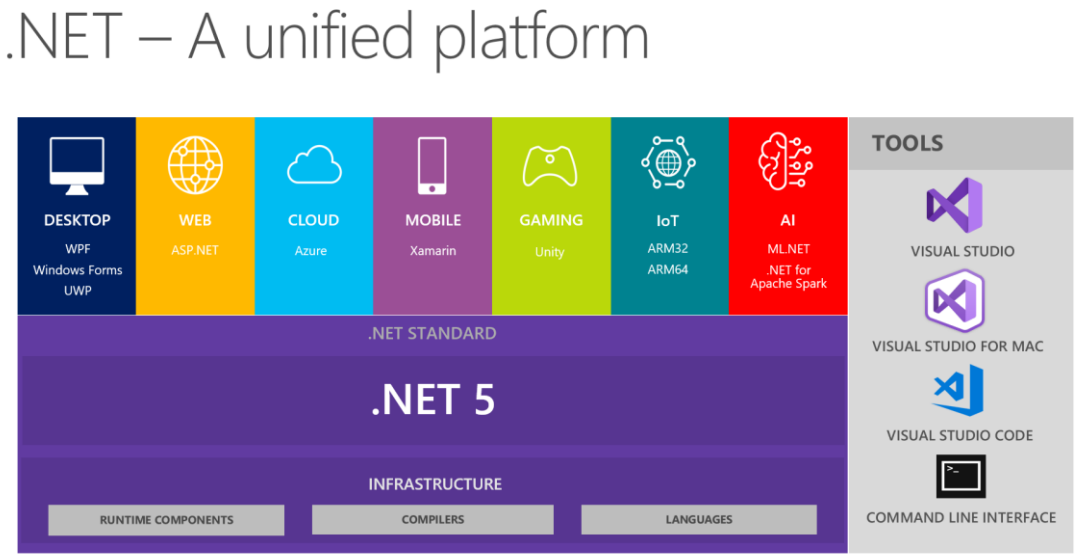


Рисунок 1.2 Платформа .NET 5

Як класичне інтегроване середовище розробки використовується Visual Studio. Для створення веб-сервісів – ASP.NET Core.

Як ORM систему для роботи з базами даних будемо використовувати – Entity Framework.

Для роботи з базами даних, хмарними технологіям та надсиланням листів використаємо відповідні відкриті пакети, що надає платформа .nuget packages.

1.2 FrontEnd

Реалізація графічного представлення сильно відрізняється від створення веб-сервісів. Тут мі будемо використовувати модульну схему, кожен модуль складається з компонентів та навігатора по ним, сервісів, а також зв'яків з іншими модулями (Рис. 1.3).

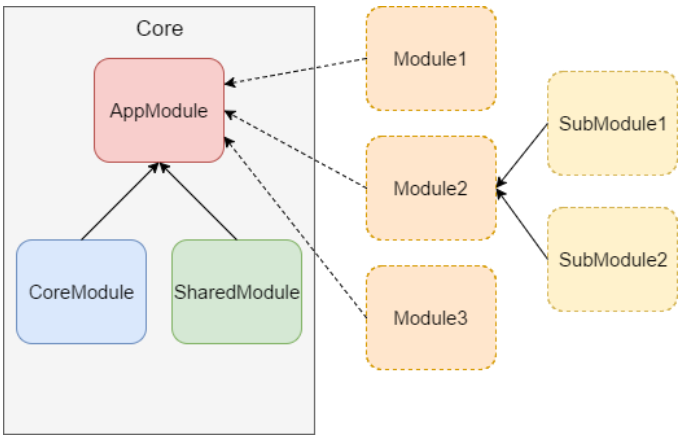


Рисунок 1.3 Структура frontend проекту

Для звернення до веб-сервісу зазвичай створюється сервіси, які використовують HTTP client.

Для наших цілей підходить frontend фреймворк Angular [3]. Дизайн буде розроблений за стандартом Material Design [1]. Для цього Angular надає спеціальний пакет Angular Material [4] з реалізованими базовими компонентами.

Як класичне інтегроване середовище розробки використовується Visual Studio Code. Зі встановленими пакетами для написання коду з використанням TypeScript, та Angular.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Onion architecture (Web API)

Термін "Onion Architecture" ("цибульна" архітектура) був запропонований Джеффрі Палермо (Jeffrey Palermo) ще в 2008 році [5]. Через роки ця концепція стала досить популярною і є однією з найбільш застосовуваних типів архітектури при побудові програми ASP.NET [6].

Onion-архітектура є поділом програми на рівні. При чому є один незалежний рівень, що знаходиться у центрі архітектури. Від цього рівня залежить другий рівень, від другого – третій і т.д.. Тобто виходить, що довкола першого незалежного рівня нашаровується другий - залежний. Навколо другого нашаровується третій, який може залежати і від першого. Образно це може бути виражене у вигляді цибулі, в якій також є серцевина, навколо якої нашаровуються всі інші шари, аж до лушпиння.

Кількість рівнів може відрізнятися, але в центрі завжди знаходиться модель домену (Domain Model), тобто класи моделей, які використовуються в додатку і об'єкти яких зберігаються в базі даних:

Перший рівень навколо моделі домену утворюють інтерфейси, що управляють роботою з моделлю домену. Зазвичай це інтерфейси репозиторіїв, якими ми взаємодіємо з базою даних.

Зовнішній рівень представляє такі компоненти, які часто змінюються. Зазвичай зовнішній рівень утворюють інтерфейс користувача, тести, якісь допоміжні класи інфраструктури програми. До цього рівня також відносяться конкретні реалізації інтерфейсів, оголошених нижче рівнях. Наприклад, реалізація інтерфейсу репозиторію, який оголошено лише на рівні Domain Services. Взагалі всі внутрішні рівні, які можна об'єднати в Application Core, визначають тільки інтерфейси, а конкретна реалізація цих інтерфейсів розташовується на зовнішньому рівні.

Також варто відзначити, що всі зовнішні сховища, як бази даних, файли, зовнішні веб-сервіси, від яких ми можемо отримувати дані, - це зовнішній по відношенню до архітектури [7].

Рівень об’єктів домену

Це центральна частина архітектури. Він містить усі об’єкти домену програми. Якщо програма розроблена за допомогою структури сутностей ORM, тоді цей рівень містить класи POCO (Code First) або Edmx (Database First) з сутностями. Ці об’єкти домену не мають жодних залежностей.

Рівень сховища

Рівень призначений для створення рівня абстракції між шаром сутностей домену та рівнем бізнес-логіки програми. Це шаблон доступу до даних, який спонукає до більш слабко пов’язаного підходу до доступу до даних. Ми створюємо загальний репозиторій, який запитує джерело даних для даних, зіставляє дані з джерела даних на бізнес-суб’єкт і зберігає зміни в бізнес-суб’єкті в джерелі даних.

Рівень сервісу

Рівень містить інтерфейси, які використовуються для зв’язку між рівнем UI та рівнем сховища. Він містить бізнес-логіку для сутності, тому його також називають рівнем бізнес-логіки.

Рівень інтерфейсу користувача

Це самий зовнішній шар. Це може бути веб-додаток, веб-API або проект модульного тестування. Цей рівень має реалізацію принципу інверсії залежностей, так що програма створює слабо пов’язану програму. Він зв’язується з внутрішнім рівнем через інтерфейси.

Переваги onion архітектури

Існує багато переваг onion архітектури, перерахованих нижче.

- Це забезпечує кращу розширюваність, оскільки всі модулі залежать від шарів або центру.
- Це забезпечує кращу можливість тестування, оскільки модульний тест можна створити для окремих шарів без впливу на інші модулів програми.
- Він створює слабко пов’язану програму, оскільки зовнішній рівень програми завжди взаємодіє з внутрішнім шаром через інтерфейси.
- Будь-яка конкретна імплементація надаватиметься додатку під час виконання.
- Сутності домену є основною та центральною частиною. Він може мати доступ як до баз даних, так і до шарів інтерфейсу користувача.
- Внутрішні шари ніколи не залежать від зовнішнього шару. Код, який міг бути змінений, повинен бути частиною зовнішнього рівня.

2.2 CQRS

CQRS означає поділ відповідальності за команди та запити - шаблон, який розділяє операції читання та оновлення для сховища даних. Впровадивши додаток CQRS, можна максимально збільшити його продуктивність, масштабованість та захист. Гнучкість, досягнута під час переходу на CQRS, дозволяє системі краще розвиватися з часом і заважає командам оновлення викликати конфлікти злиття лише на рівні домену [8].

У традиційних архітектурах аналогічна модель використовується у базах даних для виконання запитів та оновлень. Цей простий підхід чудово підходить для базових операцій створення, читання, оновлення та видалення. Але в складніших додатках такий підхід не завжди буде зручним. Наприклад, під час читання даних програма може виконувати багато різних запитів і повертати кілька об'єктів передачі різних форм. Це ускладнює зіставлення об'єктів. Для запису даних модель може використовувати складні процедури перевірки та процеси бізнес-логіки. В результаті ви отримаєте надто складну модель, яка виконує надто багато функцій.

Робочі навантаження читання та записи часто асиметричні з дуже різними вимогами до продуктивності та масштабування.

- Часто існує невідповідність між уявленнями для читання та запису, наприклад додатковими стовпцями або властивостями, які мають бути оновлені правильно, навіть якщо вони не потрібні в рамках операції.
- Змагання за дані може виникати, якщо операції виконуються паралельно з тим самим набором даних.
- Традиційний підхід може негативно вплинути на продуктивність через навантаження на сховища даних та рівень доступу до даних, а також складність запитів, необхідних для отримання інформації.

- Управління безпекою та дозволами може стати складним завданням, оскільки кожна сутність схильна до операцій читання та запису, які можуть надавати дані в неправильному контексті.

CQRS розділяє операції читання та запису в різні моделі, використовуючи команди для оновлення даних та запитів для читання даних.

- Команди мають грунтуватися на завданнях, а чи не основі даних.
- Команди можуть розміщуватись у черзі для асинхронної обробки, а не оброблятися синхронно.
- Запити ніколи не змінюють дані у базі даних. Запит повертає об'єкт передачі даних, який не містить відомостей про предметну область.

Наявність окремих моделей запитів та оновлень спрощує проектування та реалізацію. Однак одним із недоліків є те, що код CQRS не може бути автоматично створений із схеми бази даних за допомогою механізмів формування шаблонів, таких як засоби ORM.

Для додаткової ізоляції можна фізично розділити дані для читання та дані для запису. І тут у базі даних для читання можна оптимізувати схему даних те щоб максимально ефективно виконувати запити. Наприклад, у ній можна зберігати матеріалізовані уявлення даних, ніж використовувати складні операції з'єднання чи складні об'єктно-реляційні зіставлення. Можна навіть застосувати інший тип сховища даних. Наприклад, база даних для запису залишиться реляційною, а для читання ви застосуєте базу даних документів.

До переваг CQRS відносяться:

- Незалежне масштабування . CQRS дозволяє роздільно масштабувати робочі навантаження читання та запису, знижуючи ризик конфліктів блокування.
- Оптимізовані схеми даних . Для процесів читання можна застосувати схему, оптимізовану для запитів, а процесів запису — іншу схему, оптимізовану для оновлень.

- Безпека . Так буде простіше призначити до виконання операцій запису даних лише допустимі сутності домену.
- Поділ проблем . Поділ процесів читання та запису дозволяє отримати більш гнучкі та прості в обслуговуванні. Більшість складної бізнес-логіки переміститься в модель запису. Це певною мірою спростить модель читання.
- Простіші запити . Зберігаючи в базі даних для читання матеріалізоване представлення даних, ви запобігтиме використанню додатком складних з'єднань у запитах.

2.3 Ключові WebAPI .nuget пакети

2.3.1 MediatR

Пакет реалізуючий шаблон посередник. Посередник — це поведінковий патерн проектування, що дає змогу зменшити зв’язаність великої кількості класів між собою, завдяки переміщенню цих зв’язків до одного класу-посередника [9].

У додатку може буде кілька описів тієї роботи, яку потрібно виконувати (наприклад, створити запис ToDo, змінити ім'я користувача і т. д.). Ці описи MediatR називаються запитами (requests). Це звичайні класи, що реалізують інтерфейс IRequest (Рис. 2.3.1). Він є маркер-інтерфейсом без жодних членів.

```
class CreateToDoItem : IRequest<int>
{
    public string ToDoItemText { get; set; }
}
```

Рисунок 2.3.1 Приклад сутності IRequest<T> пакету MediatR

Ці класи можуть не містити жодної логіки. Вони є просто контейнери даних, необхідні виконання операцій.

Параметр T відповідає за тип, що буде повернений запитом або командою. Наприклад, при створенні запису ToDo вам може вимагати отримати ID цього запису.

У MediatR цей код для обробки запиту називається обробником запиту (request handler). Обробники запиту повинні реалізовувати інтерфейс IRequestHandler<TRequest, TResponse> (Рис. 2.3.2), де TRequestмає бути IRequest<TResponse>:

```
class CreateToDoItemHandler : IRequestHandler<CreateToDoItem, int>
{
    public Task<int> Handle(CreateToDoItem request, CancellationToken cancellationToken)
    {
        ...
    }
}
```

Рисунок 2.3.2 Приклад сутності IRequestHandler<TRequest, TResponce> пакету MediatR

Як бачите, цей інтерфейс вимагає реалізувати єдиний метод Handle, який асинхронно виконує необхідну операцію та повертає потрібний результат.

MediatR використовує контейнер залежностей. При розробці на ASP.NET Core, можна скористатися пакетом MediatR.Extensions.Microsoft.DependencyInjection.

2.3.2 Entity Framework

Entity Framework — це платформа ORM з відкритим вихідним кодом для додатків .NET, яку підтримує Microsoft. Це дозволяє розробникам працювати з даними, використовуючи об’єкти специфічних для домену класів, не зосереджуючи увагу на базових таблицях і стовпцях бази даних, де ці дані зберігаються. За допомогою Entity Framework розробники можуть працювати на більш високому рівні абстракції, коли вони мають справу з даними, а також можуть створювати та підтримувати орієнтовані на дані програми з меншою кількістю коду в порівнянні з традиційними програмами.

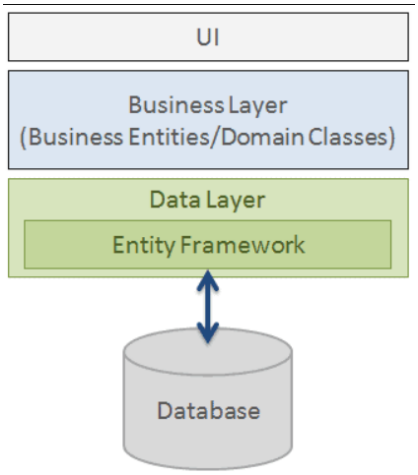


Рисунок 2.3.3 Entity Framework

Наступний Рисунок (2.3.3) ілюструє, де Entity Framework використовується у програмах.

Особливості Entity Framework:

- Міжплатформенність: EF Core – це кросплатформна платформа, яка може працювати на Windows, Linux і Mac.
- Моделювання: EF (Entity Framework) створює EDM (модель даних об’єкта) на основі об’єктів POCO (звичайний старий об’єкт CLR) із властивостями отримання/встановлення різних типів даних. Він використовує цю модель під час запиту або збереження даних сутності в базовій базі даних.
- Запити: EF дозволяє нам використовувати запити LINQ [10](C#/VB.NET) для отримання даних із базової бази даних. Постачальник бази даних перекладе ці запити LINQ на мову запитів,

специфічну для бази даних (наприклад, SQL для реляційної бази даних). EF також дозволяє нам виконувати необроблені запити SQL безпосередньо до бази даних.

- Відстеження змін: EF відстежує зміни, що відбулися в екземплярах ваших сутностей (значення властивостей), які потрібно надіслати до бази даних.
- Збереження: EF виконує команди INSERT, UPDATE та DELETE до бази даних на основі змін, що відбулися у ваших об’єктах під час виклику SaveChanges()методу. EF також надає асинхронний SaveChangesAsync()метод.
- Паралелізм: EF використовує Optimistic Concurrency за замовчуванням для захисту від перезапису змін, внесених іншим користувачем після того, як дані були отримані з бази даних.
- Транзакції: EF виконує автоматичне керування транзакціями під час запиту або збереження даних. Він також надає можливості для налаштування управління транзакціями.
- Кешування: EF включає перший рівень кешування з коробки. Таким чином, повторні запити повертатимуть дані з кешу замість того, щоб перейти до бази даних.
- Вбудовані умови: EF дотримується конвенцій щодо шаблону програмування конфігурації та включає набір правил за замовчуванням, які автоматично конфігурують модель EF.
- Конфігурації: EF дозволяє нам налаштувати модель EF за допомогою атрибутів анотації даних або Fluent API для заміни умовних умов.
- Міграції: EF надає набір команд міграції, які можна виконати на консолі диспетчера пакетів NuGet або в інтерфейсі командного рядка для створення або керування базовою схемою бази даних.

2.3.3. Dapper

Dapper є технологією зіставлення (мапінгу) результатів sql-запитів з класами c#. У цьому плані Dapper трохи схожий на Entity Framework. У той же час за рахунок своєї легковаги Dapper забезпечує більшу продуктивність і швидше дозволяє виконувати запити, ніж Entity Framework.

Для здійснення запитів Dapper надає для об'єктів IDbConnection метод розширення Query<T> [11], який як параметр приймає SQL-вираз і може повертати об'єкт типу T, з яким зіставляються результати запиту.

Завдяки більшій продуктивності ніж у Entity Framework, Dapper чудово підходить для виконання громіздких запитів на отримання даних з базі даних, тому ідеально підходить для оброблення Query команд в архітектурі CQRS.

2.4 JWT Authentication

JSON Web Token (JWT)— це JSON об'єкт, визначений у відкритому стандарті RFC 7519. Він вважається одним із безпечних способів передачі інформації між двома учасниками. Для створення необхідно визначити заголовок (header) із загальною інформацією по токену, корисні дані (payload), такі як id користувача, його роль тощо. та підписи (signature) [12].

Додаток використовує JWT для перевірки аутентифікації користувача таким чином:

- Спочатку користувач заходить на сервер аутентифікації за допомогою аутентифікаційного ключа (це може бути пара логін/пароль , або Facebook ключ, або Google ключ, або ключ від іншого облікового запису).
- Потім сервер автентифікації створює JWT та відправляє його користувачеві.
- Коли користувач робить запит до програми API, він додає до нього отриманий раніше JWT .
- Коли користувач робить API запит, програма може перевірити за переданим із запитом JWT чи є користувач тим, за кого себе видає. У цій схемі сервер програми налаштований так, що зможе перевірити, чи є вхідний JWT саме тим, що був створений сервером аутентифікації (процес перевірки буде пояснений пізніше більш детально).

Для безпеки також потрібно додавати Refresh Token.

Токени доступу (JWT) – це токени, за допомогою яких можна отримати доступ до захищених ресурсів. Вони короткоживучі , але багаторазові . У них може бути додаткова інформація, наприклад, час життя або IP-адреса, звідки йде запит. Все залежить від бажання розробника.

Рефреш токен (RT) - ці токени виконують лише одне специфічне завдання - отримання нового токена доступу. І цього разу без сервера авторизації не обійтись. Вони довгоживучі , але одноразові .

Основний сценарій використання такий: як тільки старий JWTспливає, то з ним ми вже не можемо отримати приватні дані, тоді відправляємо RT і нам приходить нова пара JWT+RT. З новим JWTми знову можемо звертатися до приватних ресурсів. Звичайно, рефреш токен теж може сплинути, але трапиться це не скоро, оскільки живе він набагато довше за свого побратима.

2.5 Docker

Docker – це програмна платформа для швидкої розробки, тестування та розгортання програм. Docker упаковує програмне забезпечення в стандартизовані блоки, які називаються контейнерами . Кожен контейнер включає все необхідне роботи програми: бібліотеки, системні інструменти, код і середовище виконання. Завдяки Docker можна швидко розгортати та масштабувати програми в будь-якому середовищі та зберігати впевненість у тому, що код працюватиме. Зручно використовувати для розгортання баз даних та різних сервісів як от Elasticsearch.

2.6 MS SQL, MongoDB

Перш за все розберемо поняття релаційних та нереляційних баз даних. Термін " реляційний " прийшов з алгебри (теорія множин). У форматі БД це, що дані реляційних баз зберігаються як таблиць і рядків. Нереляційні бази даних розміщують інформацію в колекціях документів JSON.

Реляційні БД використовують мову SQL (структурованих запитів). Структура таких баз даних дозволяє пов'язувати інформацію з різних таблиць за допомогою зовнішніх ключів (або індексів), які використовуються для унікальної ідентифікації будь-якого фрагмента атомарного даних в цій таблиці. Інші таблиці можуть посилатися цей зовнішній ключ, щоб створити зв'язок між частинами даних і частиною, яку вказує зовнішній ключ.

Навіщо потрібні нереляційні БД? Їхня головна перевага — високий рівень безпеки та можливість обійти апаратні обмеження (за допомогою Sharding).

SQL підійде, якщо потрібна обробка великої кількості складних запитів або рутинного аналізу даних. Вибирайте реляційну БД , якщо потрібна надійна обробка транзакцій та цілісність посилань.

Якщо обсяг даних великий, краще використовувати NoSQL. Відсутність структурованих механізмів прискорить процес обробки Big Data. А ще це безпечніше – такі БД складніше зламати.

Вибирайте NoSQL , якщо:

- необхідно зберігати масиви в об'єктах JSON;
- записи зберігаються у колекції з різними полями чи атрибутами;
- необхідно горизонтальне масштабування.

Microsoft SQL Server - це відмінний варіант реляційної БД. Діалект T-SQL обробляє процедури, вбудовані функції та змінні. Є важливе обмеження: Microsoft SQL Server працюватиме лише з Linux або Windows. Простий

інтерфейс прискорить процес міграції БД, якщо ви користувалися іншою системою.

MongoDB - це якісний безкоштовний продукт, який найчастіше використовують під час роботи з NoSQL. Рішення дозволяє змінювати схеми даних у процесі роботи, масштабуватись по горизонталі.

2.7 ElasticSearch

Використовується для швидкого пошуку по тегах. Elasticsearch дозволяє виконувати та комбінувати багато типів пошуку — структурований, неструктурований, географічний, метричний — будь-яким способом. Elasticsearch - це розподілений пошуковий та аналітичний двигун на базі Apache Lucene. Незабаром після випуску в 2010 році Elasticsearch стала найпопулярнішим пошуковим движком і зазвичай використовується для таких прикладів як аналіз журналів, повнотекстовий пошук, інтелектуальні системи безпеки, бізнес-аналітика та моніторинг поточних процесів.

Ви можете надсилати дані до Elasticsearch у вигляді документів JSON за допомогою API або інструментів прийому, таких як Logstash та Amazon Kinesis Firehose. Elasticsearch автоматично зберігає вихідний документ і додає посилання на нього до індексу кластера, включаючи можливість пошуку. Потім можна знайти і витягти документ, використовуючи API Elasticsearch. Також для візуалізації даних та створення інтерактивних панелей управління можна задіяти Kibana – інструмент візуалізації з Elasticsearch [13].

2.8 AWS Textract

Amazon Textract – сервіс машинного навчання (ML), який автоматично витягує друкований та рукописний текст та дані із сканованих документів. Цей процес виходить за рамки простого оптичного розпізнавання символів (OCR) і дозволяє ідентифікувати, розуміти та вилучати дані з форм та таблиць. Сьогодні багато компаній отримують дані із сканованих документів (наприклад, PDF-файлів, зображень, таблиць та форм) вручну або за допомогою простого програмного забезпечення для оптичного розпізнавання тексту, якому потрібне ручне налаштування і, часто, оновлення при зміні форми. Щоб позбавитися проблем дорогої ручної обробки, Textract читає та обробляє будь-які типи документів за допомогою машинного навчання, просто і точно виймаючи друкований і рукописний текст, таблиці та інші дані [14]. Незалежно від того, Чи автоматизуєте ви процес видачі кредитів або витягуєте дані з рахунків та чеків, можна швидко налаштувати автоматичну обробку документів та приймати рішення на основі отриманої інформації. Textract здатний витягти дані всього за кілька хвилин замість годин або днів. Крім того, за допомогою Amazon Augmented AI можна додати в процес перевірки, що виконуються співробітниками, щоб контролювати роботу моделей і звіряти конфіденційні дані.

2.9 AWS S3 AND SNS

Amazon Simple Storage Service (Amazon S3) – це сервіс зберігання об'єктів, що пропонує найкращі в галузі показники продуктивності, масштабованості, доступності та безпеки даних. Клієнти будь-якої величини та з будь-якої промислової галузі можуть зберігати та захищати необхідний обсяг даних для практично будь-якого прикладу використання. Наприклад, для озер даних, хмарних програм та мобільних програм. Вигідні класи сховища та прості у використанні інструменти адміністрування дозволяють оптимізувати витрати, організувати дані та точно налаштувати обмеження доступу відповідно до потреб бізнесу чи законодавчих вимог.

Зручний у використанні разом з іншими сервісами AWS. Наприклад при обробці файлів Textract попередньо завантажуючи їх до S3.

Amazon SNS — це служба керованих повідомлень, яка дає змогу відокремити publishers від subscribers. Це корисно для обміну повідомленнями від програми до програми для мікросервісів, розподілених систем і програм без сервера.

2.10 AWS Comprehend

Amazon Comprehend – це сервіс обробки природної мови (NLP), у якому для пошуку цінної інформації та взаємозв'язків у тексті застосовуються технології машинного навчання.

Amazon Comprehend використовує попередньо навчену модель для вивчення та аналізу документа або набору документів, щоб отримати інформацію про нього. Ця модель постійно навчається на великому об’ємі тексту, тому вам не потрібно надавати навчальні дані [15].

Amazon Comprehend збирає наступні типи інформації:

- Об’єкти – посилання на імена людей, місць, предметів і місць, що містяться в документі.
- Ключові фрази – фрази, які з’являються в документі. Наприклад, документ про баскетбольний матч може повертати назви команд, назву місця проведення та остаточний рахунок.

- Особиста інформація (PII) – персональні дані, які можуть ідентифікувати особу, наприклад адреса, номер банківського рахунку або номер телефону.
- Мова – домінуюча мова документа.
- Настрої – домінуючий настрій документа, який може бути позитивним, нейтральним, негативним або змішаним.
- Цільовий настрій – настрої, пов’язані з конкретними об’єктами в документі. Настрої щодо кожної сутності можуть бути позитивними, негативними, нейтральними або змішаними.
- Синтаксис – частини мови для кожного слова в документі.

2.11 ML Named Entity Recognition

Іменована сутність - це слово або словосполучення, що обозначає предмет або явища певної категорії. Прикладами іменованих сутностей є імена людей, назви організацій та локацій [16]. Завдання розпізнавання іменованих сутностей (Named Entity Recognition, NER) полягає у виділенні та класифікація іменованих сутностей у тексті (Рис 2.11.1). За одним завданням NER, насправді, стоїть дві:

1) виявити, що якась послідовність слів це іменована сутність;

2) зрозуміти, до якого класу (ім'я людини, назва організації, місто тощо.) ця іменована сутність належить.

Рисунок 2.11.1 результат Named Entity Recognition

Для нашої задачі потрібно буде розпізнавати інформацію з резюме апліканта. А саме повне ім’я, електронну скриньку, досвід та уміння.

РОЗДІЛ 3. ІДЕЯ ТА СУТНОСТІ ПРОЄКТУ

Ідея проєкту – це створення веб-додатку для автоматизації процесів найму, відбору та централізації інформації про працівників з використанням машинного навчання та хмарних технологій.

У системі існують користувачі – HR’и деякої компанії.

У компанії є створені певні проєкти, які зараз розробляються. До проєкту буде можливість створити вакансію з певними вимогами та етапами відбору. Проходження по етапам викликає автоматизовані дії, які заощаджують час менеджера, та збільшують його продуктивність.

Наприклад є можливість створювати шаблони для листів на пошту і потім автоматично надсилати їх коли кандидат приходить, або покидає даний етап. Також можливе автоматичне створення відео зустрічей, або додавання завдань для менеджера у його власний список.

Система буде мати аплікант– це потенційний працівник, дані про якого, його досвід, контакти та навички будуть зберігатися у системі. Кожен апплікант може бути поданим на будь яку вакансію, що створені у системі, при подачі апліканта до певної вакансії, він стає кандидатом. У системі будуть існувати багато шляхів отримання інфорації про аппліканта, починаючи від мануального додавання, закінчуючи діставанням інформації з резюме за допомогою машинного навчання та хмарних технологій.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ BACKEND`У (ONION АРХИТЕКТУРА)

4.1 Рівні

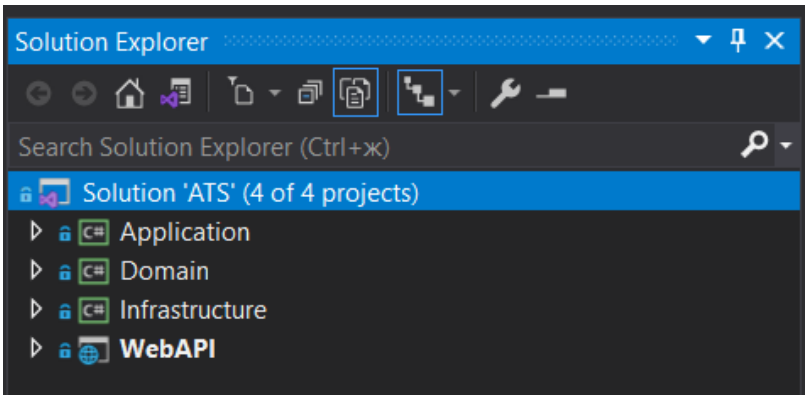


Рисунок 4.1.1 Структура backend`у

Усього в проєкті є 3 рівня Domain → Infrastructure, Application → WebAPI (Рис. 4.1.1).

Центральним є Domain, який містить класи моделей, інтерфейси, та базові сутності. За ним йде рівень Infrastructure та Application. Infrastructure реалізує усі команди та запити відповідно до архітектури моделі CQRS, використовуючи інтерфейси репозиторій з домену, Application в свою чергу реалізує усі репозиторії які потім будуть використовуватись Infrastructure за допомогою dependency injection. Зовшнішнім шаром є проект WebAPI, який реалізую принцип інверсії залежностей та буде використовуватись користувачем (Frontend`ом).

4.2 Domain

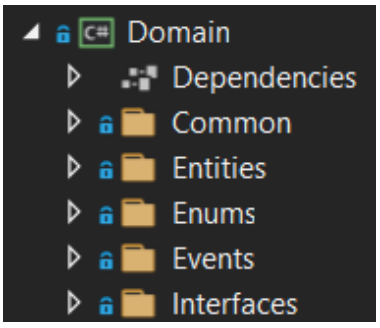


Рисунок 4.2.1 Структура Domain

Домейн складається з фолдеру Common з базовими сутностями. Entites з POCO об’єктами усього додатку.

Enums з усіми перелічувальними типами. Events з сутностями подій.

Та Interfaces з інтерфейсами усіх репозиторіїв які реалізуються в Infrastructure. (Рис. 4.2.1)

4.3 Infrastructure

На цьому шарі виконується підключення до існуючих в додатку баз даних та хмарних технологій (Рис. 4.3.1):

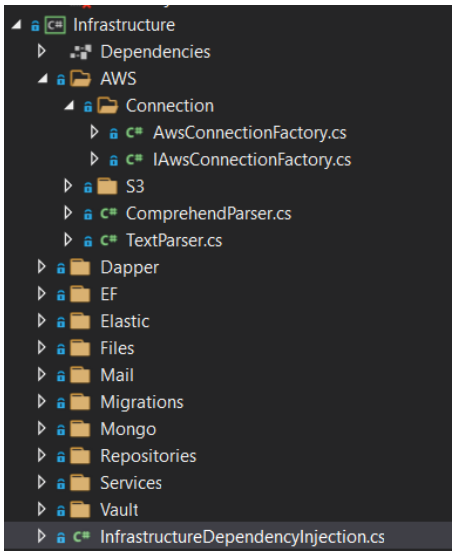


Рисунок 4.3.1 Структура Infrastructure

- Перш за все підключення до MS SQL та модель контексту. Це ключова реляційна база даних в якій зберігаються усі дані окрім файлів та шаблонів повідомлень. Тобто уся відносно невелика інформація.
- Підключення до MongoDB, яка використовується для збереження шаблонів повідомлень, та файлі, що зберігаються в системі.
- Підключення до сервісів AWS, а саме AWS Comprehend для розпізнавання іменованих сутностей, AWS Textract для розпізнавання тексту в картинках та пдф файлах, і також AWS S3 в якому зберігаються усі файли, що використовуються сервісами Амазону, резюме аплікантів, та усі зображення.

Цей рівень реалізує усі репозиторії читання/запису які зазначені у Домейні. Для репозиторіїв запису використовується Entity Framework, так як він дуже зручний у використанні. В свою чергу для репозиторіїв читання ми використовуємо Dapper, який використовую sql код, для покращення продуктивності.

І останнє за, що відповідає цей рівень, це інверсію залежностей. В ньому визначається залежність усіх класів до відповідних інтерфейсів (Рис. 4.3.2).

```
private static IServiceCollection AddWriteRepositories(this IServiceCollection services)
{
    services.AddScoped<IWriteRepository<User>, WriteRepository<User>>();
    services.AddScoped<IWriteRepository<Vacancy>, WriteRepository<Vacancy>>();
    services.AddScoped<IWriteRepository<Project>, WriteRepository<Project>>();
    services.AddScoped<IWriteRepository<Company>, WriteRepository<Company>>();
    services.AddScoped<IWriteRepository<Stage>, WriteRepository<Stage>>();
    services.AddScoped<IWriteRepository<RefreshToken>, WriteRepository<RefreshToken>>();
    services.AddScoped<IWriteRepository<ApplyToken>, WriteRepository<ApplyToken>>();
    services.AddScoped<IWriteRepository<Role>, WriteRepository<Role>>();
    services.AddScoped<IWriteRepository<UserToRole>, WriteRepository<UserToRole>>();
```

Рисунок 4.3.2 Реалізація інверсії залежностей

4.4 Application

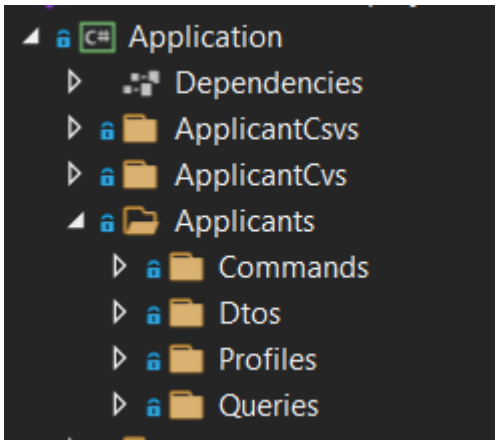


Рисунок 4.4.1 Структура Application

Реалізує усі команди та запити (Архітектура CQRS). Команди та запити розміщені у папках по контексту разом з DTO об’єктами, що їх стосуються (Рис. 4.4.1).

Data Transfer Object (DTO) — один із шаблонів проєктування, який використовують для передачі даних між підсистемами програми. Ці об`єкти наділу будуть використовуватись для спілкування з клієнтом, серіалізуючись у JSON. DTO реалізується для того, щоб не створювати прямої залежності між домейном та зовнішнім слоєм.

Також для коректного мапингу DTO об’єктів до сутностей реалізуються Профілі, які указують як саме потрібно приводити один об’єкт до іншого.

Запит реалізується за допомогою двух класів, перший з яких є самою сутністю запиту з переліком усіх необхідних для виконання команди полів. І

```
public class GetUserByIdQuery : IRequest<UserDto>
{
    2 references
    public string Id { get; }

    1 reference
    public GetUserByIdQuery(string id)
    {
        Id = id;
    }
}
```

Рисунок 4.4.2 Приклад реалізації IRequest<T>

який наслідується від інтерфейсу IRequest<T> пакету MediatR (Рис. 4.4.2). Другий клас є хендлером запиту, який реалізує інтерфейс IRequestHandler тієї ж бібліотеки (Рис. 4.4.3). За допомогою dependency injection він отримує необхідні репозиторії, виконує певну задачу та повертає тип зазначений при створенні запиту.

```
public class GetUserByIdQueryHandler : IRequestHandler<GetUserByIdQuery, UserDto>
{
    protected readonly IUserReadRepository _repository;

    protected readonly IMapper _mapper;
    protected readonly ISender _mediator;

    0 references
    public GetUserByIdQueryHandler(IUserReadRepository repository, IMapper mapper,
     ISender mediator)
    {
        _repository = repository;
        _mapper = mapper;
        _mediator = mediator;
    }

    0 references
    public async Task<UserDto> Handle(GetUserByIdQuery query, CancellationToken _)
    {
        var user = await _repository.GetByIdAsync(query.Id);
        return _mapper.Map<UserDto>(user);
    }
}
```

Рисунок 4.4.3 Приклад реалізації IRequestHandler

4.5 WebAPI

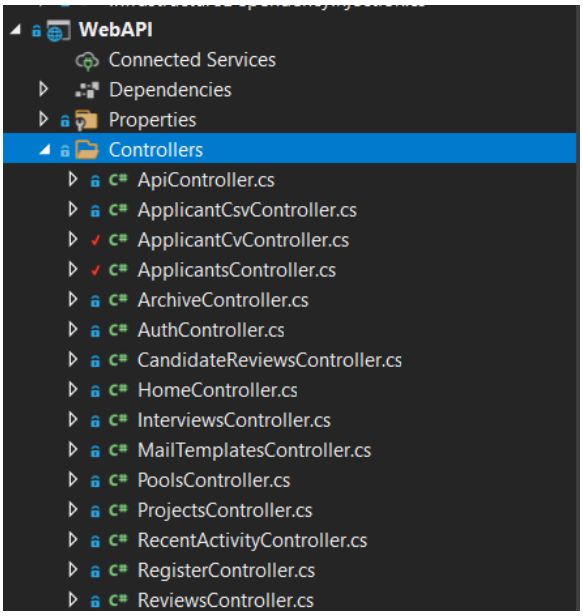


Рисунок 4.5.1 Структура WebAPI

Зовнішній рівень, який реалізую контролери та являється інтерфейсом для спілкування з користувачем (Frontend`ом) (Рис. 4.5.1). Усі ендпоінти використовують MediatR для надсилання команд та запитів, які виконують певну задачу.

Також ВебАпи реалізує деякі middleware. Проміжне програмне забезпечення — це програмне забезпечення, яке надає загальні послуги та можливості додаткам за межами того, що пропонує операційна система. Керування даними , послуги додатків, обмін повідомленнями, аутентифікація та керування API зазвичай обробляються проміжним програмним забезпеченням. Вони забезпечують пре та пост обробку запитів.

Перший middleware “AddAuthHeaderOperationFilter” забезпечує додавання логін-токена у випадку, якщо користувач залогінений і ендпоінт до якого він звернувся потребує даних користувача, перед виконанням запиту.

Другий middleware “ ErrorHandlerMiddleware” відловлює усі помилки спровоковані у додатку, та залежно від статус коду, та додаткової інформації надає користувачу інформацію про помилку при виконанні, при цьому ми не повинні засмічувати код try-catch блоками, так як за це відповідає цей проміжний модуль.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ FRONTENDУ (Angular)

Рисунок 4.1 Лого Angular

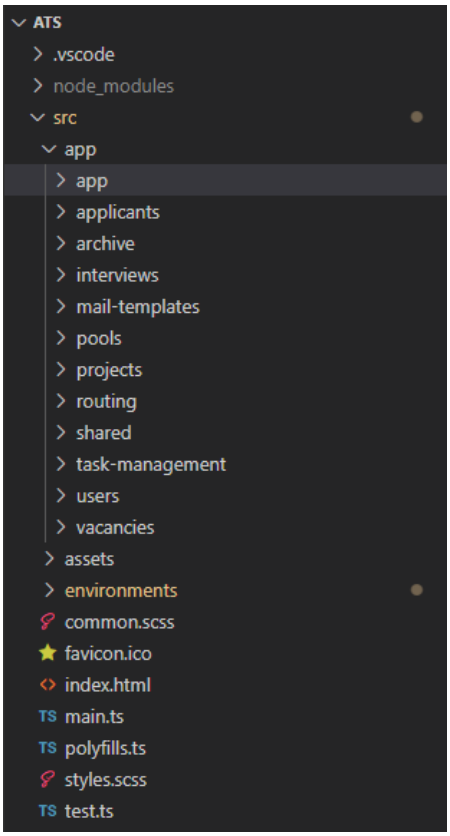


Рисунок 4.2 Структура frontend`у

Angular — це платформа JavaScript з відкритим кодом, написана на TypeScript [17] (Рис. 4.1). Google підтримує його, і його основна мета — розробляти SPA програми. Як фреймворк, Angular має очевидні переваги, а також надає стандартну структуру для роботи розробників. Це дозволяє користувачам створювати великі програми в обслуговуваний спосіб.

Односторінковий додаток (SPA – Single Page Application) — це веб-додаток або веб -сайт , який взаємодіє з користувачем шляхом динамічного переписування поточної веб-сторінки з новими даними з веб-сервера замість стандартного методу веб-браузера , який завантажує цілі нові сторінки. Метою є швидші переходи, завдяки чому веб-сайт буде більше схожий на нативну програму.

Проєкт складається з папок app, assets та environments. Assests містить усі матеріали розміщені на веб-додатку, усі лого, заставки та бекграунди (Рис. 4.2). Environments відповідає за налаштування різних середовищ. І ключова папка app містить у собі всі модулі програми.

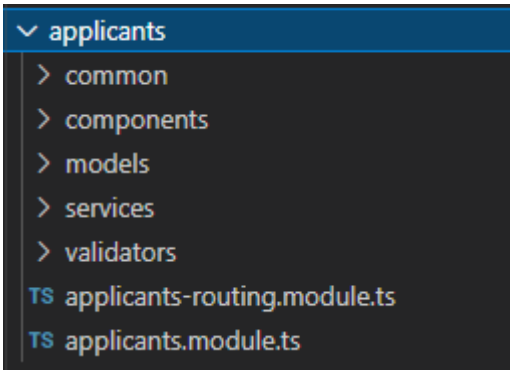


Рисунок 4.3 Вигляд модулю applicants

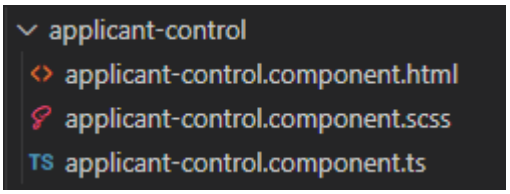


Рисунок 4.4 Вигляд компоненту

Кожен модуль містить у собі бібліотеки, які він імпортує, налаштований навігатор, який по шляху визначає який компонент мусить бути використаним для відображення інформації. Перелік моделей які використовуються в межах цього модулю. А також сервіси та безліч компонентів (Рис. 4.3).

Компоненти – це трійка файлів (у випадку наявності тестів четвірка) (Рис.4.4). Html файл – в ньому міститься структура та зовнішній вигляд сторінки. Scss файл – sass скриптова метамова, яка інтерпретується в css, відповідає за стилі компонента. Typescript файл – в ньому описані методи (логіка) пов’язана з компонентом.

Сервіси – це сутності які відповідяють за спілкування з backend`ом. Для цього сервіс використовую HttpClient за допомогою якого можливо здійснення запитів до ендпоінтів різних контролерів (Рис. 4.5).

```
@Injectable({
  providedIn: 'root',
})
export class SelfApplyService {

  constructor(private httpService: HttpClientService) { }

  public sendApplyConfirmEmail(applyInfo: ApplyTokenInfo): Observable<void> {
    return this.httpService.postRequest<void>('/SelfApply/email-confirm-apply', applyInfo);
  }

  public MarkTokenUsed(token: string): Observable<void> {
    return this.httpService.postRequest<void>(`/SelfApply/mark-token-used/${token}`, new Object());
  }

  public getApplyConfirmEmailToken(email: string, vacancyId: string): Observable<string[]> {
    return this.httpService
      .getRequest<string[]>(`/SelfApply/email-confirm-apply/${vacancyId}/${email}`);
```

Рисунок 4.5 Приклад сервісу

РОЗДІЛ 5. РЕЗУЛЬТАТ (ВЗАЄМОДІЯ З ВЕБ-ДОДАТКОМ)

5.1 Сторінка авторизації

Початкова сторінка, з якої користувач не може потрапити до додатку, поки не введе емайл та пароль. Також є можливість відновити пароль, якщо ви його забули (Рис. 5.1.1).

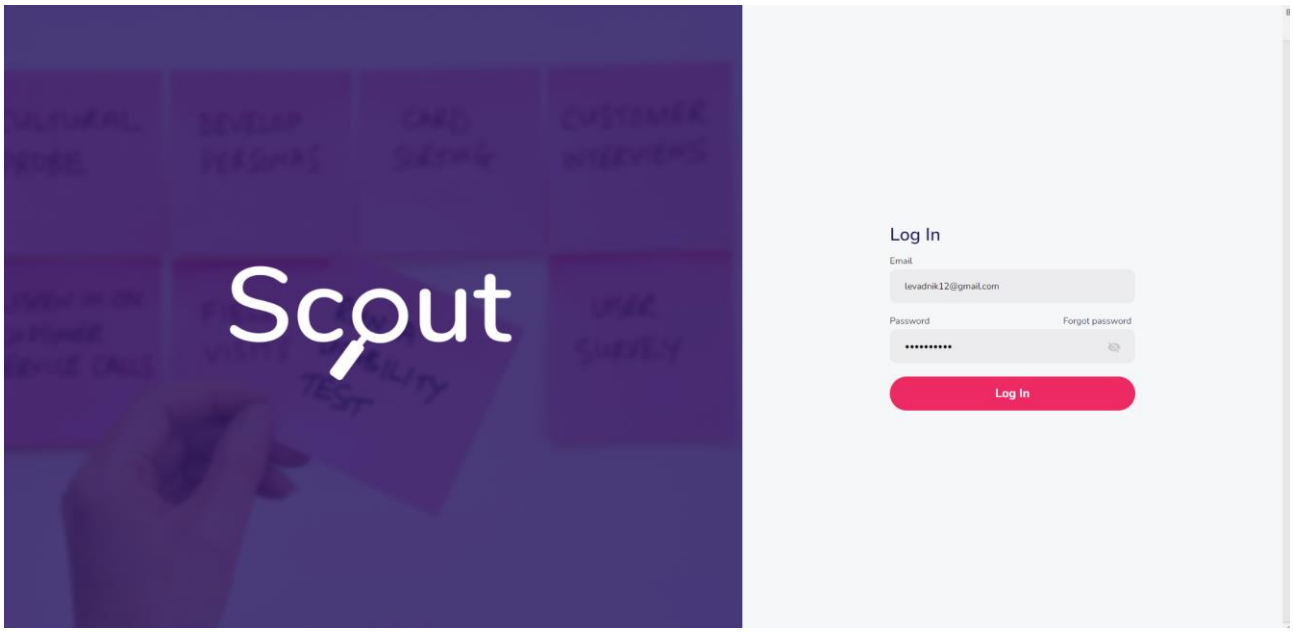


Рисунок 5.1.1 Сторінка авторизації

5.2 Навігація, аккаунт та ролі в системі

У системи є три можливих ролі користувача:

- Неавторизований користувач. Користувач який ще не авторизувався в системи, тоді він може потрапити лише на сторінку авторизація, та на сторінку подання заяви до вакансії, у випадку, якщо ця людина аплікант.
- HR. Користувач який має аккаунт у системі, прив’язаний до первої компанії, він має доступ до усіх функцій та сторінок, окрім управління аккаунтів у системі, та можливості видаляти проекти та вакансії.

- HR Lead. Користувач який мая доступ до усіх сторінок та функціоналу первної компанії.

Сторінки (Рис.5.2.1):

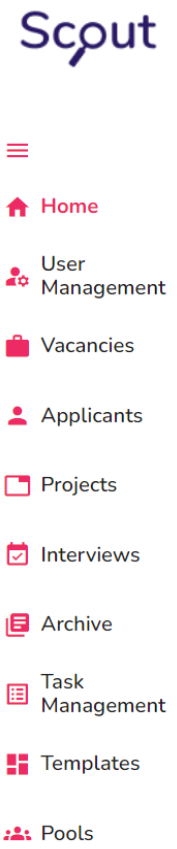


Рисунок 5.2.1 Навігація

Home – Інформаційна сторінка на якій зібрана найбільш важлива інформація

User Management – Сторінка яка є доступною лише HR Lead`у. на цій сторінці можлово додавати/видаляти користувачів.

Vacancies – Сторінка з переліком вакансій, функціями додавання/редактування/видалення/налаштування вакансій, переходом на канбан дошку, посиланнями на форму само подання до позиції.

Applicants – Сторінка з переліком апплікантів системи, функціями додавання/редактування/видалення апплікантів.

Projects – Сторінка з переліком проєктів системи, функціями додавання/редактування/видалення проєктів.

Interviews – Сторінка у вигляді календаря, з відображенням відео зустрічей, та можливістю створення нових.

Archive – Сторінка до якої потрапляють проєкти та вакансії після архівування. Також сторінка з якої можливо зробити повне видалення цих сутностей, але тільки під роллю HR Lead`а.

Task Management – Сторінка с картками-завданнями HR`а, які можуть створюватись автоматично, або ж самим HR`ом.

Templates – Сторінка з переліком шаблонів повідомлень, функціями додавання/редактування/видалення шаблонів.

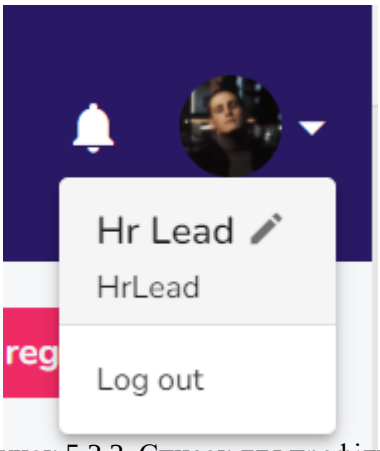


Рисунок 5.2.2 Список для профілю

Pools – Сторінка з переліком пулів апплікантів, під пулом мається на увазі групи апплікантів (HR може групувати апплкантів для власних цілей, по досвіду, технологіям, або ж знанню англійської, а потім додавати одразу пулл кандидатів на нові вакансії), функціями додавання/редактування/видалення пулів.

Також у правому верхньому кутку на будь-якій сторінці ви можете побачити фото аккаунту, та натиснувши кнопки для зміни інформації про себе, або ж виходу с системи (Рис 5.2.2).

При редагуванні аккаунту (Рис. 5.2.3) ми можемо змінити фото/ім`я/прізвище/…

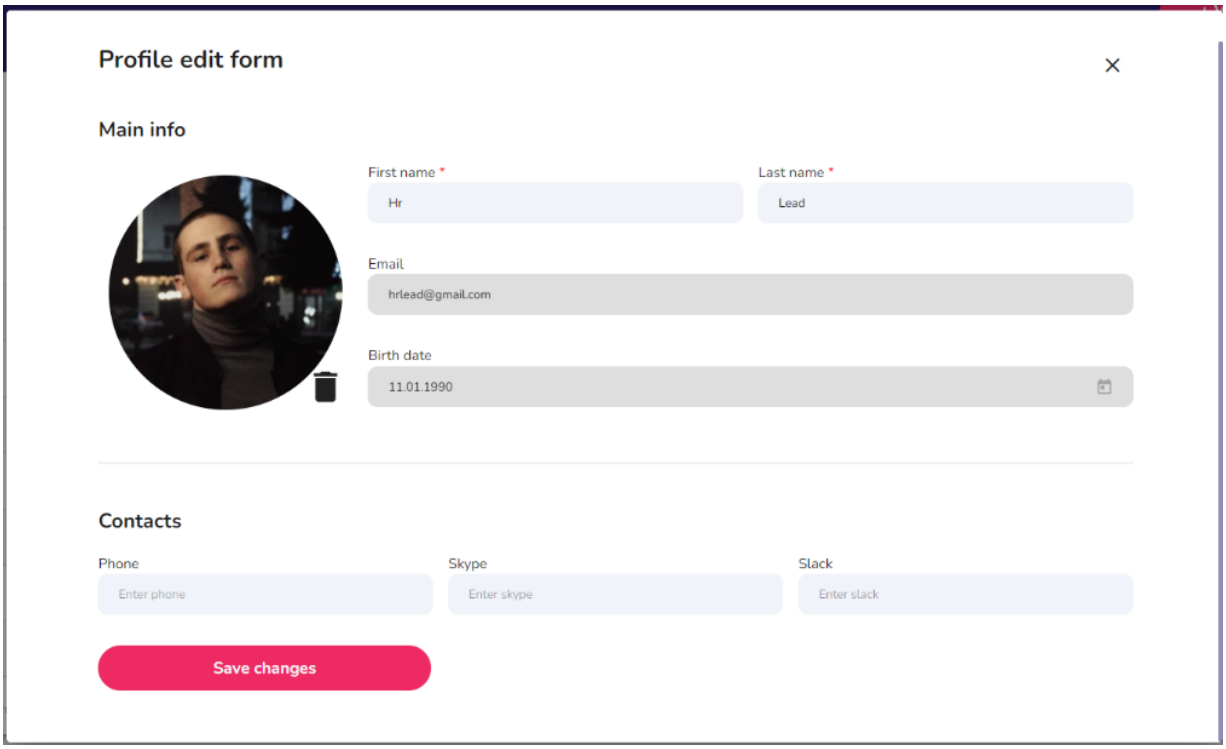


Рисунок 5.2.3 Редагування аккаунту

5.3 Сторінка Home

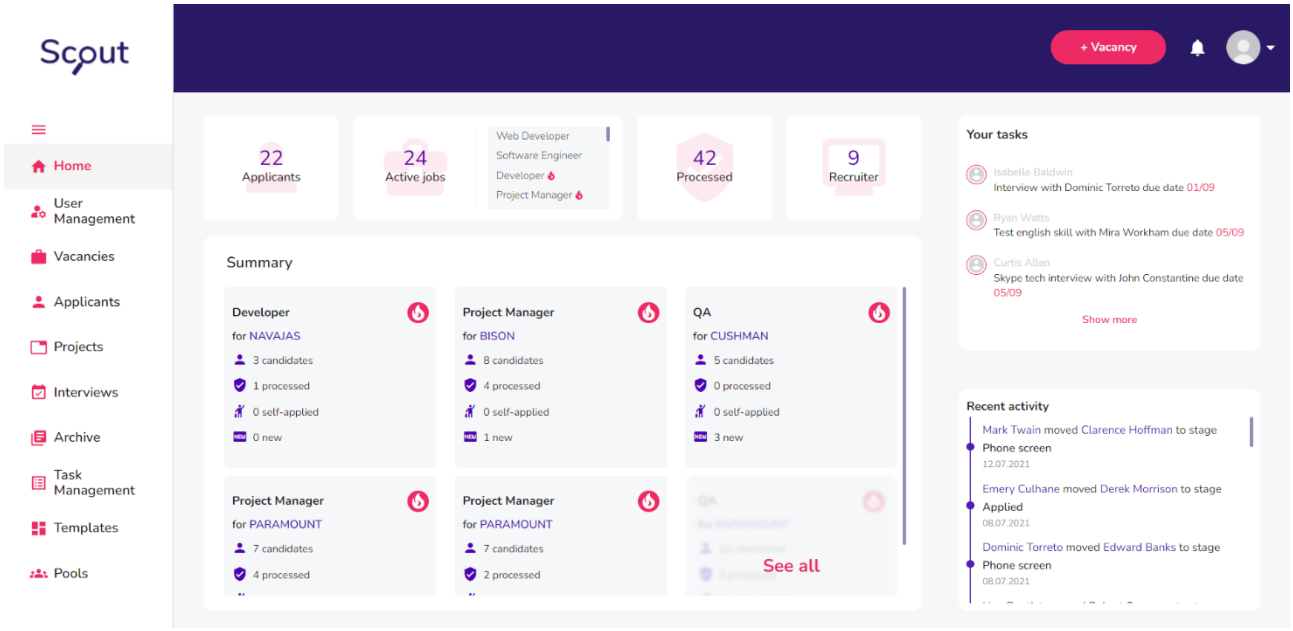


Рисунок 5.3.1 Сторінка Home

Тут можна побачити основну інформацію (Рис. 5.3.1), для того щоб швидко зорієнтуватись по статистиці компанії (Кількість апплікантів в системі, активних вакансій, кандидатів котрі дійшли до останньої стадії відбору та кількість рекрутерів). А також навігувати по гарячих вакансіях. Справа є блок тасок та остання активність що стосується процесу найму, пересуванню кандидатів по етапах тої чи іншої вакансії. Також ми можемо мінімізувати навігаційне меню.

5.4 Глобальні стилі та Сторінка User Management

Увесь додаток розроблено по шаблону Material Design [1]. Використовується однакова палітра кольорів, та однаковий стиль оформлення.

Також фактично на кожній сторінці є можливість обирати улюблені проекти/вакансії/… та бачити відповідно обране на вкладках All/Followed. Усі таблиці реалізують пагинацію (розбиття данних на сторінки, можливість змінити кількість відображення даних).

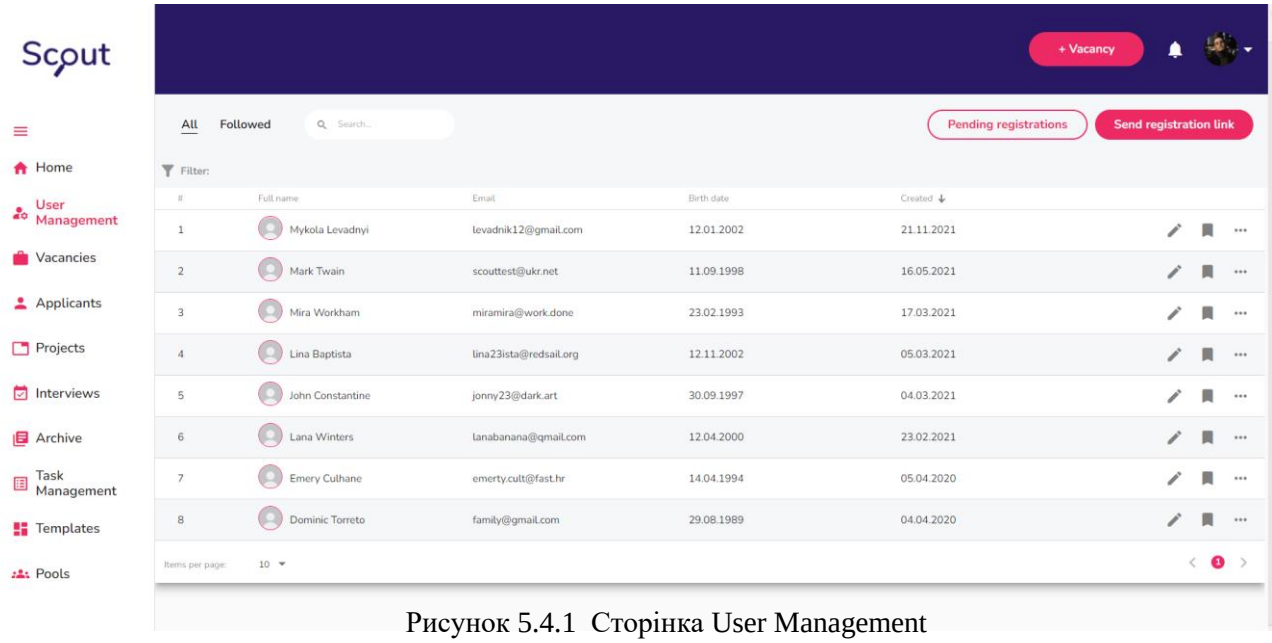


Рисунок 5.4.1 Сторінка User Management

На сторінці менеджменту (Рис. 5.4.1) користувачів ми можемо видаляти існуючих (сценарій покидання рекрутером роботи). Або надсилати посилання для регістрації для нових користувачів (Рис. 5.4.2) і відстежувати стан реєстрації уже надісланих посилань (Рис. 5.4.3).

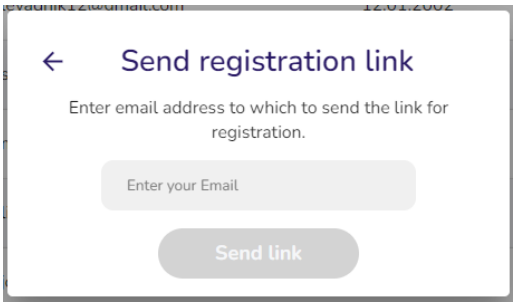


Рисунок 5.4.2 Надіслання запрошеня

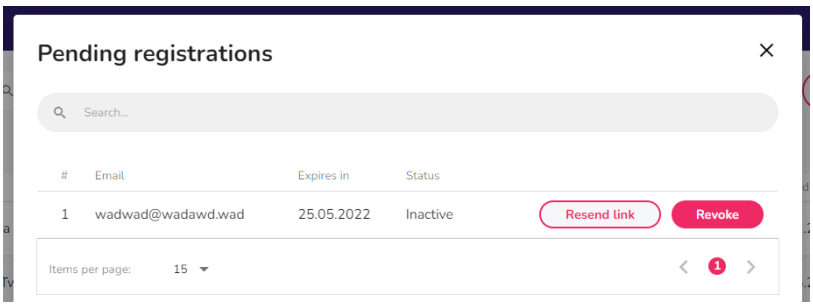


Рисунок 5.4.3 Список надійсланих запрощешь

5.5 Сторінка Projects

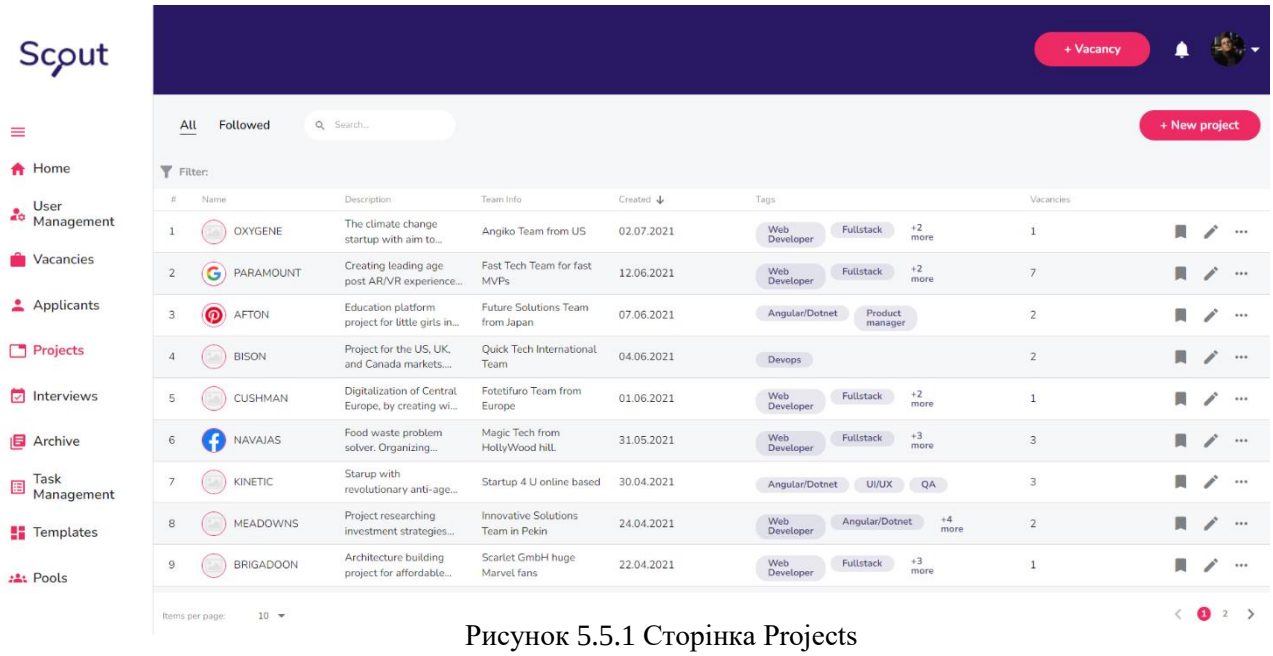


Рисунок 5.5.1 Сторінка Projects

Тут ми можемо побачити основну інформацію та функції взаємодії з проектами (Рис. 5.5.1).

5.6 Сторінка Vacancies

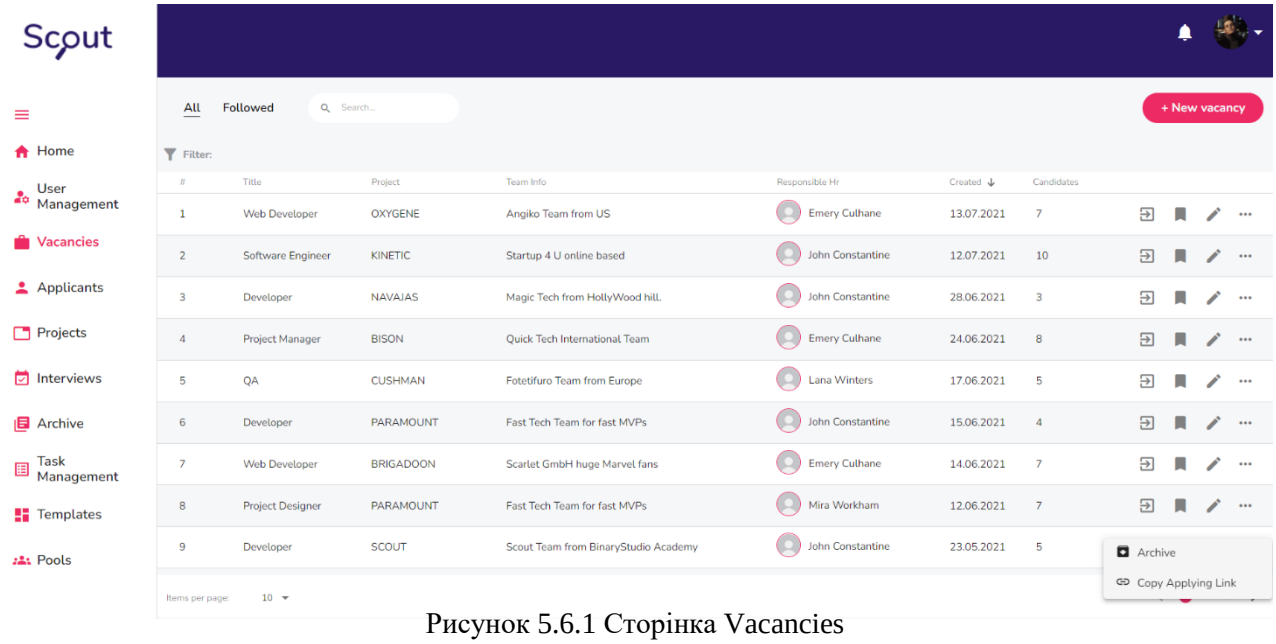


Рисунок 5.6.1 Сторінка Vacancies

Тут ми можемо побачити основну інформацію та функції взаємодії з вакансіями (Рис. 5.6.1). Також є можливість копіювати посилання на форму самоподання та можливо перейти до канбан дошки вакансії (до цих пунктів ми повернемось у розділі).

Вакансія є однією з основних сутностей в системі, тому зупинимось на формі створення/редагування вакансії. Для початку потрібно задати Назву, опис, вимоги, обрати проект, рівень зарплати, який саме працівник потрібен, додати посилання на сайти де вакансія опублікована, та додати перелічінь тегів (Рис.5.6.2).

Але головним налаштуванням є етапи (вокрфлоу) вакансії (Рис. 5.6.3). Тут ми задаємо умовні стадії по яким повинен пройти кандидат, щоб буди найнятим. Також для цих етапів реалізована автоматизація для рекрутерів.

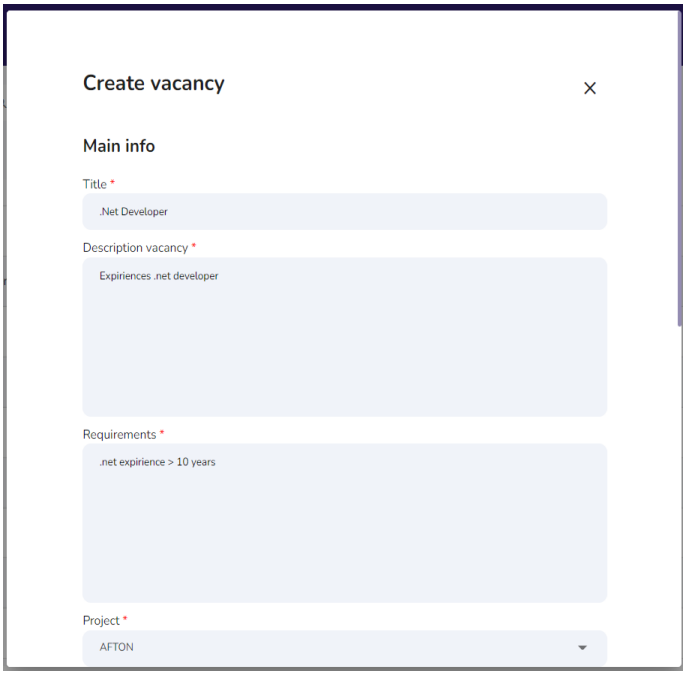


Рисунок 5.6.2 Створення вакансії

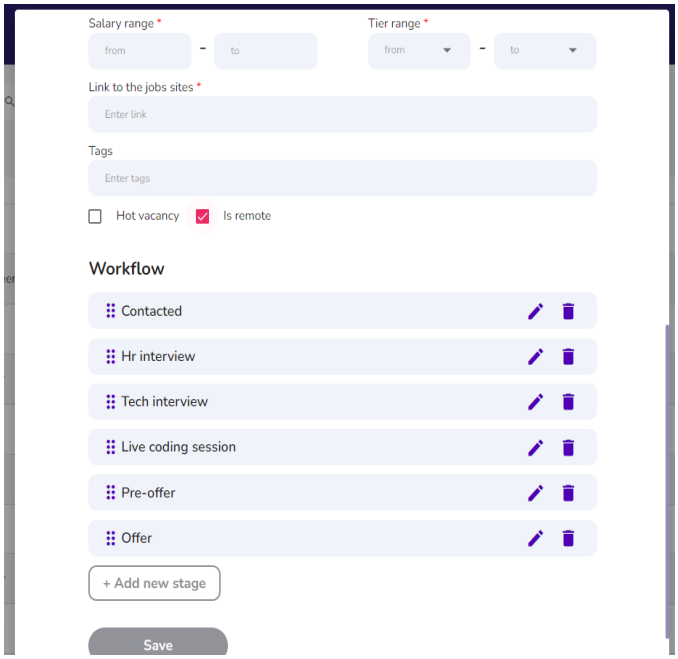


Рисунок 5.6.3 Етапи вакансії

При редагуванні етапу ми можемо указати дію при приєднанні кандидата до цього етапу і при покиданні (Рис. 5.6.4).

Існує три типи дій:

- Add Task – автоматично створює завдання для менеджера, яке відображається на сторінці завдань (див. розділ 5.10)
- Shedule interview – автоматично назначає інтервью з кандидатом, яке відображається на сторінці інтервью (див. розділ 5.9)

- Send mail – надає можливість автоматично відсилати повідомлення на пошту аппліканту, з підготовленим шаблоном (зі сторінки шаблонів див розділ 5.11).

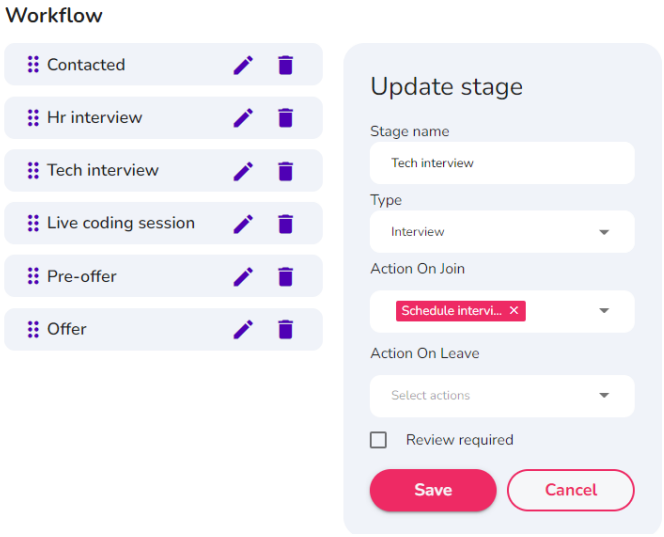


Рисунок 5.6.4 Налаштування етапа вакансії

У такий спосіб рекрутер може економити собі дуже багато часу, встигати та не забувати важливих речей.

5.7 Сторінка Applicants

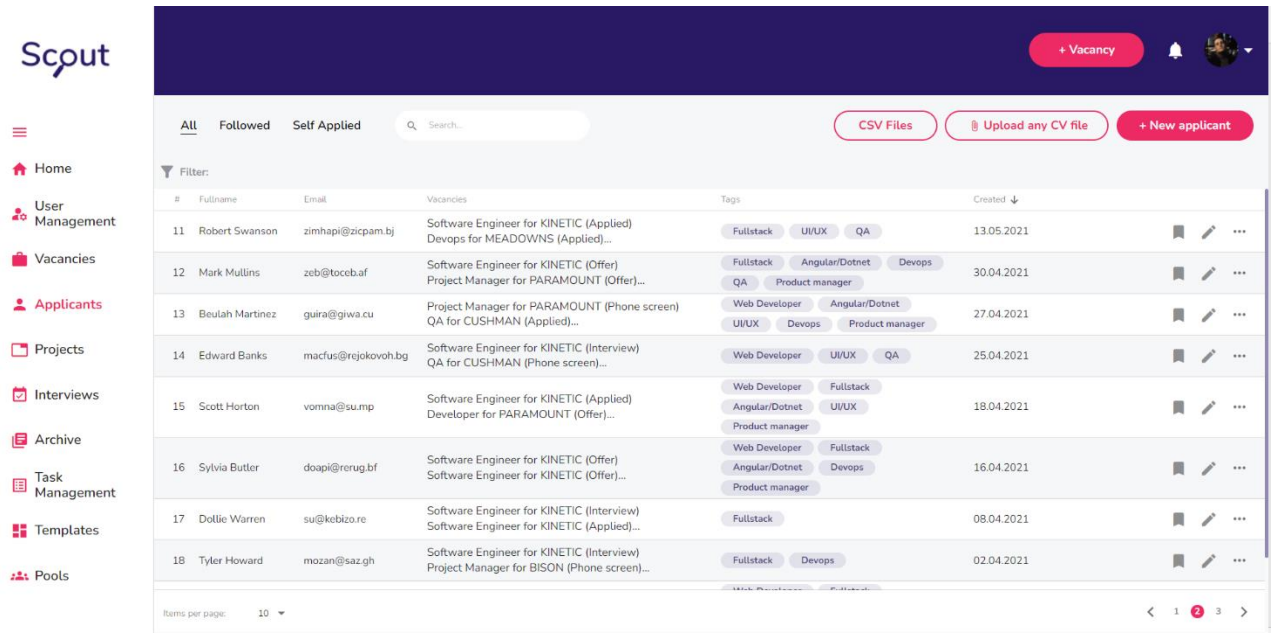


Рисунок 5.7.1 Сторінка Applicants

На цій сторінці можна побачити усіх апплікантів системи, їх уміння, та вакансії на яких вони є кандидатами (Рис. 5.7.1). Більш детально обговоримо варіанти додавання аппліканта до системи, наразі наша система має бальшу кількість варіантів додавання апплікантів до системи, ніж у інших сервісів:

5.7.1 Мануальне додавання

Рекрутер має можливість самостійно додати аппліканта до системи, заповнивши форму (Рис. 5.7.2).

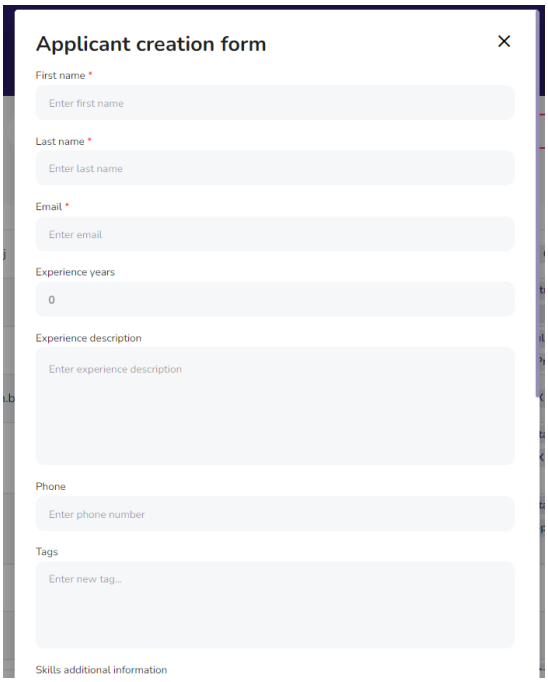


Рисунок 5.7.2 Форма додавання аппліканта

5.7.2 Chrome Extension

Додатково за допомогою мови JavaScript було розроблене розширення, яке дозволяє дійстати інформацію з популярної соцмережі для пошуку вакансій

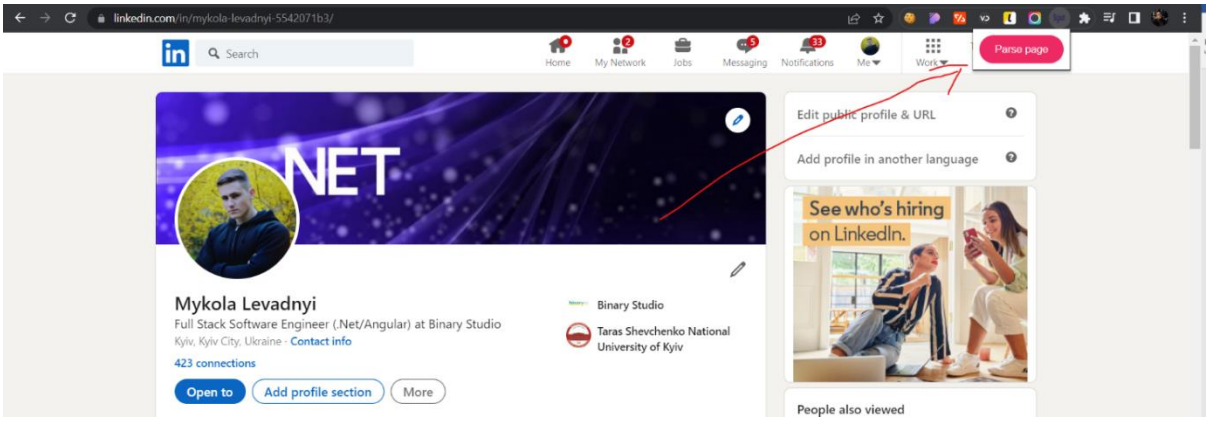


Рисунок 5.7.3 Вигляд розширення google chrome

Linkedin (Рис. 5.7.3). Користувач може завантажити розширення і потім на сторінці Linkedin він зможе дістати інформацію аппліканта (Рис. 5.7.4).

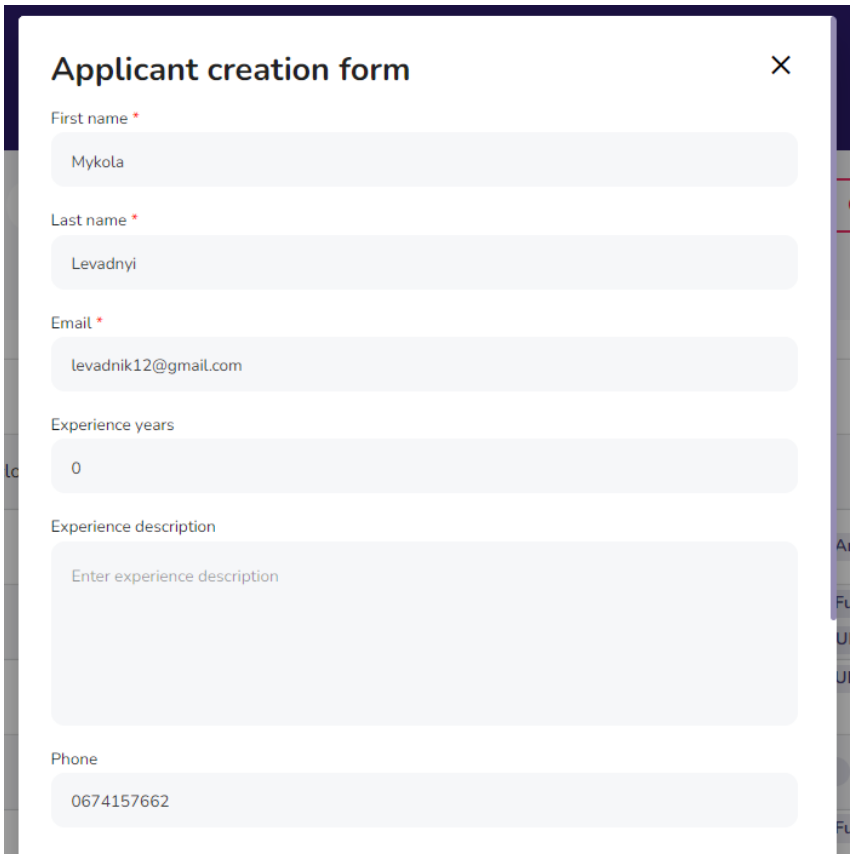


Рисунок 5.7.4 Форма заповнена даними зі сторінки Linkedin

Це допомагає HR`у економити час та не дублювати інформацію власноруч до системи.

5.7.3 Upload from csv files

Зазвичай у HR`ів є власні файли в яких вони зберігають інформацію про кандидатів. Популярним форматом файлу для цього є .csv. Фактично від представляє інформацію у види таблиці, де дані кожного рядка розділяються комою. Для цього в системі також є механізм зчитування. Рекрутер може завантажити .csv файл, після чого від побачить інтерфейс для додавання аплікантів з файлу (Рис. 5.7.4).

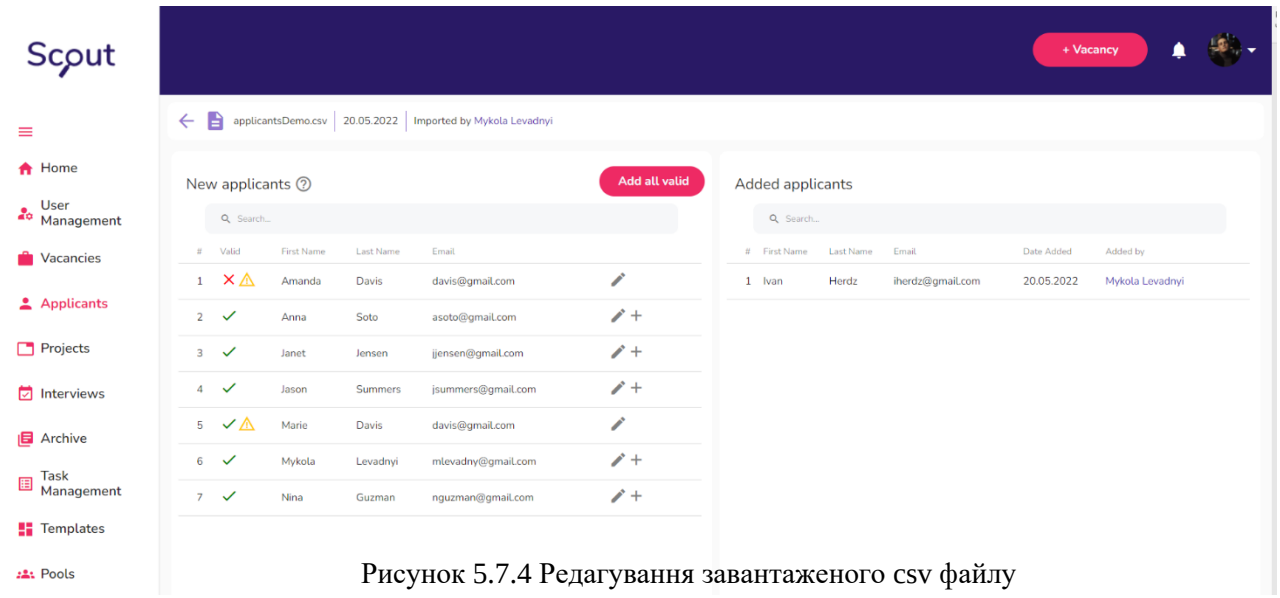


Рисунок 5.7.4 Редагування завантаженого csv файлу

У апліканта з файлу можуть бути різні стани, валідний – якщо всі дані є і його можна додати в систему, невалідний – якщо чогось бракує, з повторюваним емейлом, тобто порушується унікальність аппліканта і він уже існую у системі.

5.7.4 Форма самоподання для кандидатів

Часто рекрутери публікують оголошення на різних сайтах (Linkedin, DOU, HeadHunter) для пошуку працівників. Але при цьому кандидати повинні написати листа, рекрутер повинен відповісти. В системі передбачена форма самоподання яку може заповнити кандидат, перейшовши за посиланням з сайту з вакансіями (скопіювати посилання можна зі сторінки вакансій) (Рис. 5.7.5).

Для початку кандидат заповнить свою пошту.

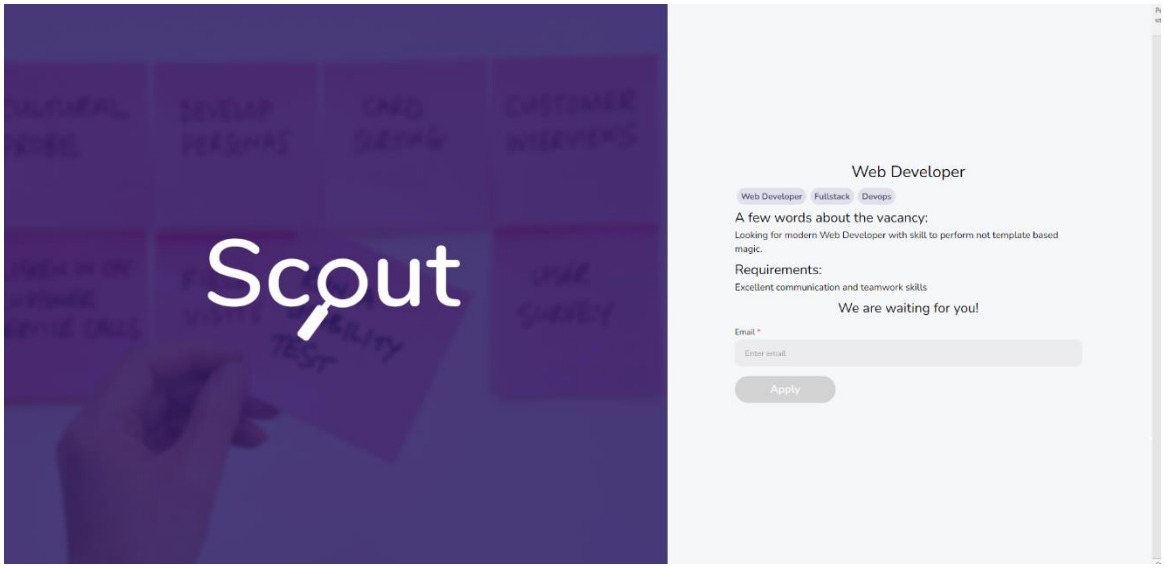


Рисунок 5.7.5 Сторінка самоподання аппліканта

Після цього йому на пошту надійде повідомлення з посиланням для продовження подання (Рис. 5.7.6).

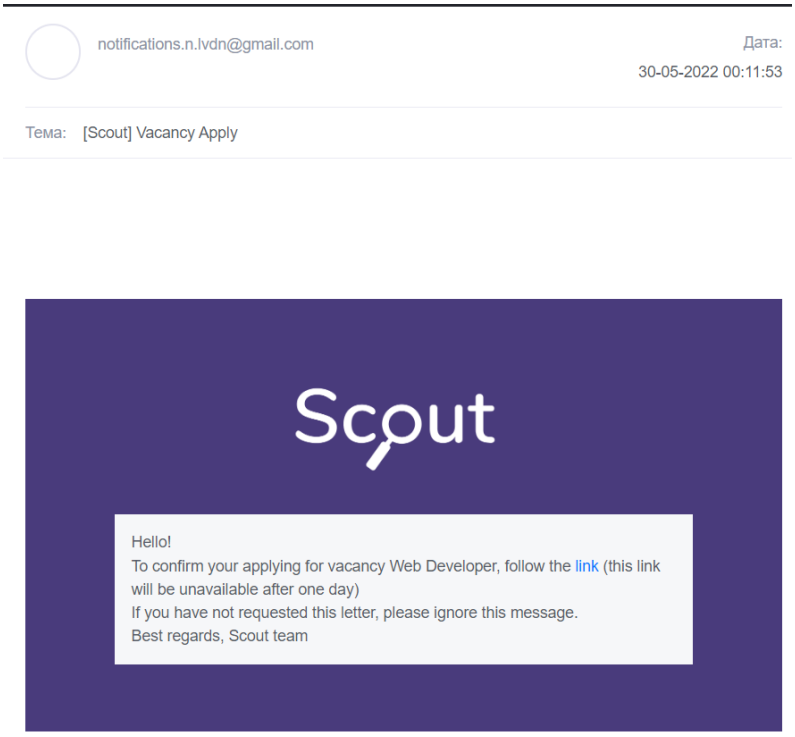


Рисунок 5.7.6 Лист про подання на вакансію

Це забезпечує захист від дудос атак, щоб людина дійсно мала емейл з якого вона намагається податись на вакансію. Потім кадидат дозаповнює форму, та надсилає дані (Рис. 5.7.7). Він з’явиться у системі з відміткою самоподаного. Також він буде поданий на вакансію, але для подібних

кандидатів автоматично з’явиться етап перет тими, що налаштовуються в вакансії, цей етап буде містити лише автоматично доданих, так HR зможе вирішити хто може перейти на початковий етап, а хто не вартий уваги.

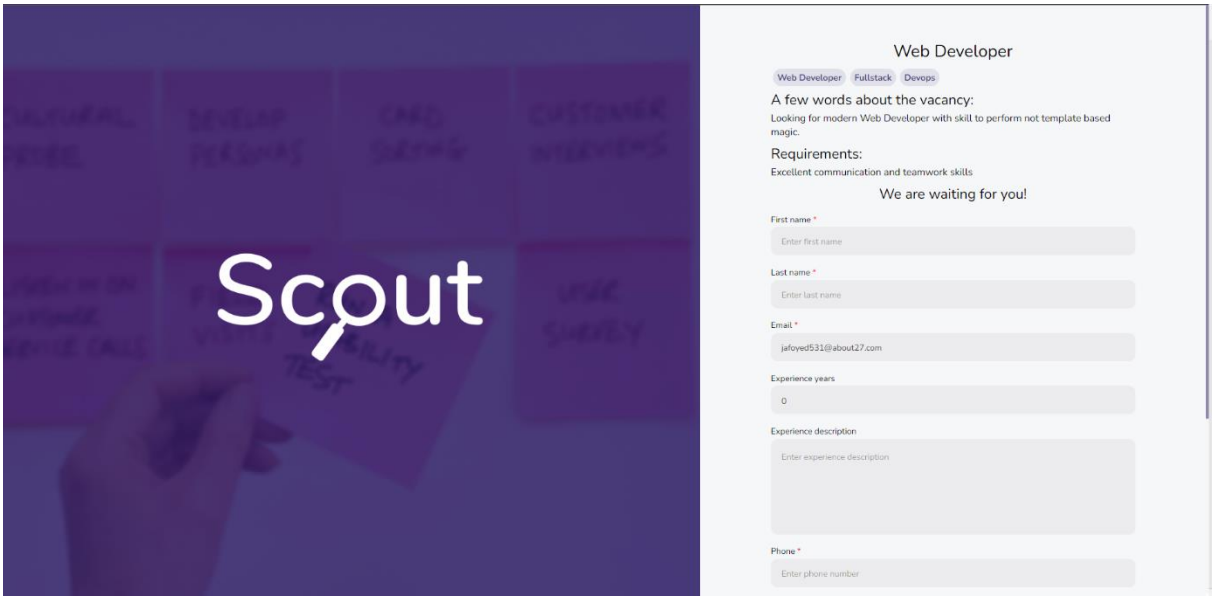
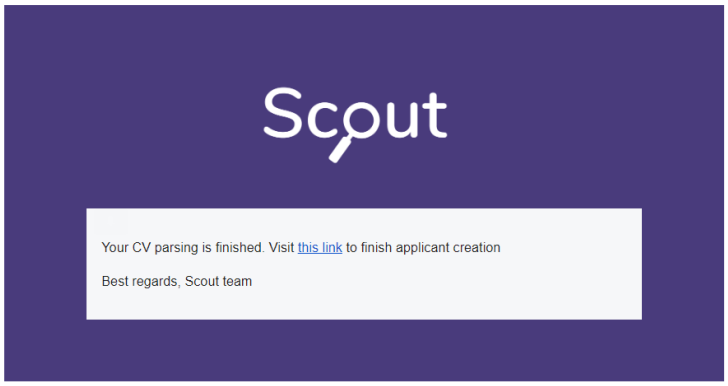


Рисунок 5.7.7 Фінальна форма самоподання

5.7.5 Парсинг резюме кандидата за допомогою машинного навчання

(Детальний опис навчання моделі див. розділ 6) У разі наявності резюме кандидата (Рис. 5.7.10) рекрутер може використати навчену модель для додавання в систему каждидата, все що потрібно це завантажити файл на сторінці апплікантів. Після чого через декілька хвилин на пошту прийде лист з



посиланням (Рис. 5.7.8). Рисунок 5.7.8 Лист опрацьований пдф

При переході на по посиланню рекрутер побачить поля які вийшло розпізнати моделі (Рис. 5.7.9), та у нього буде можливість мануально вибрати, що додати. Після чого він може створити кандидата.

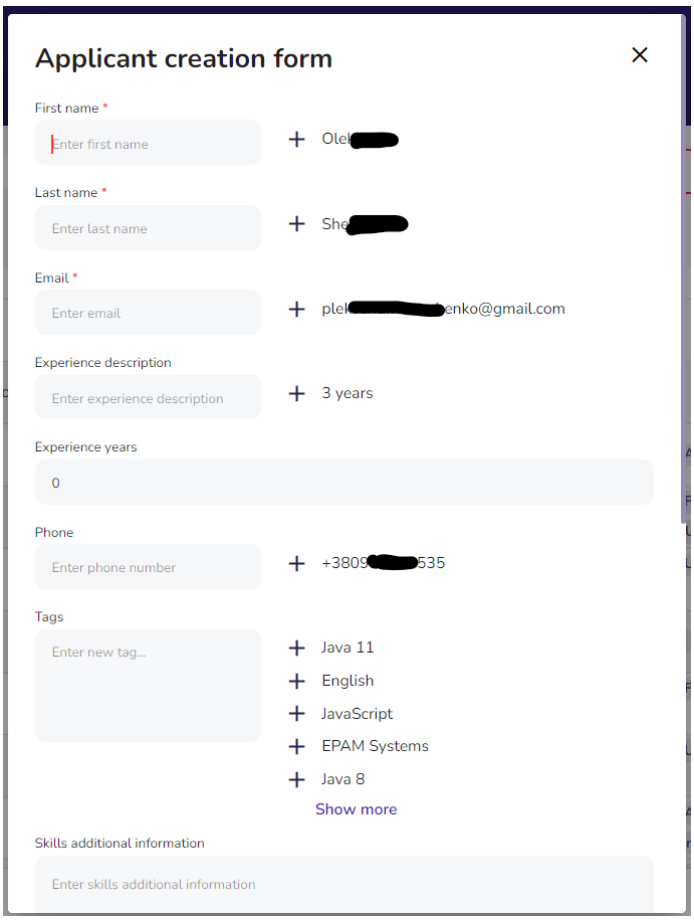


Рисунок 5.7.9 Результат опрацювання пдф

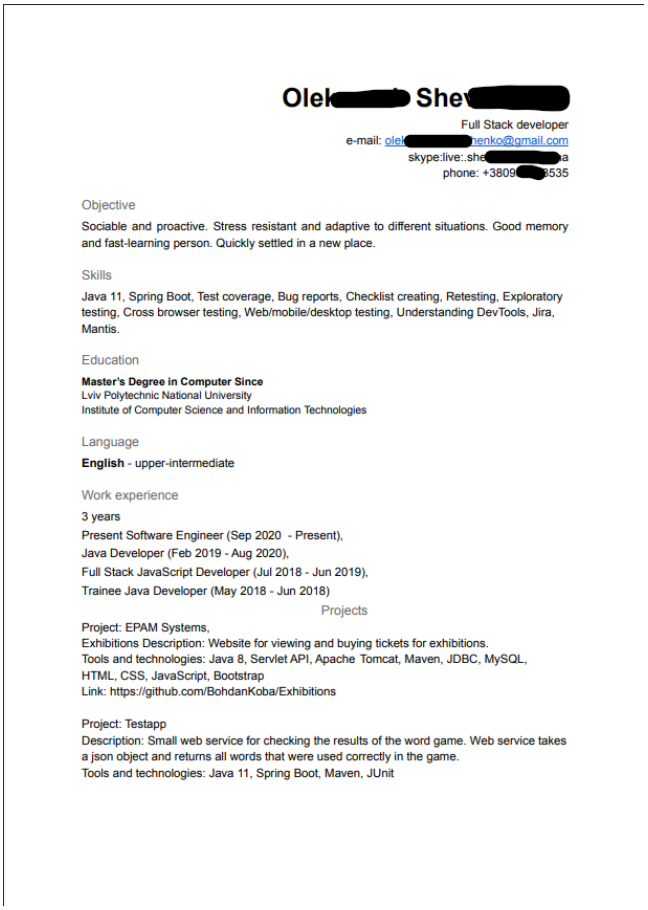


Рисунок 5.7.10 Опрацьований пфд

5.8 Канбан дошка вакансії

При переході на канбан дошку вакансії зі сторінки вакансії ми побачимо усі етапи вакансії та її кандидатів (Рис. 5.8.1).

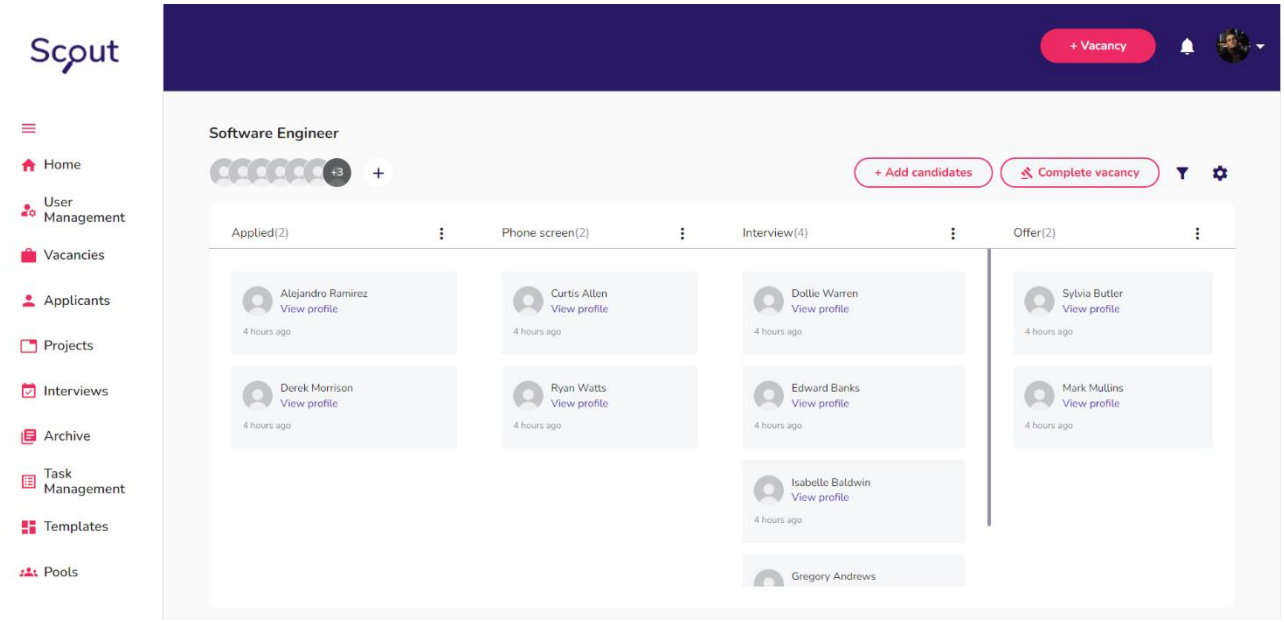


Рисунок 5.8.1 Канбан дошка вакансії

Перехід кандидата на наступний етап виконується перетаскуванням за допомогою миші. Також ми маємо можливість додавати нових кандидатів (Рис. 5.8.2) та дивитися детальну інформацію по кожному кандидату (Рис. 5.8.3).

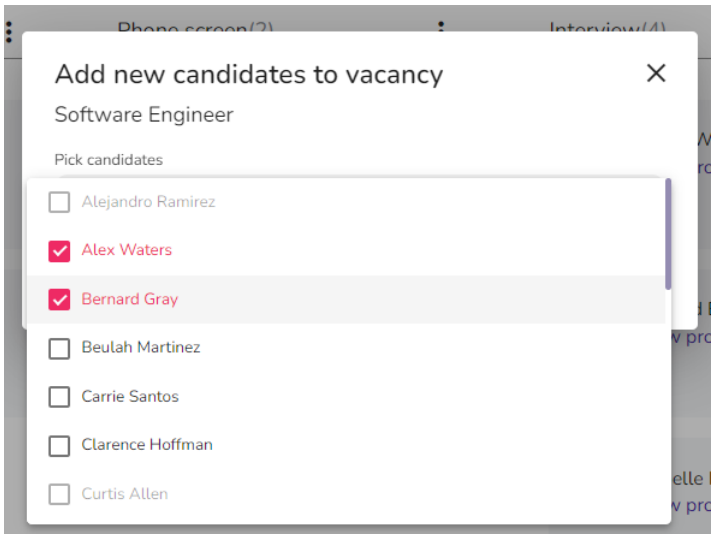


Рисунок 5.8.2 Додавання кандидатів на вакансію

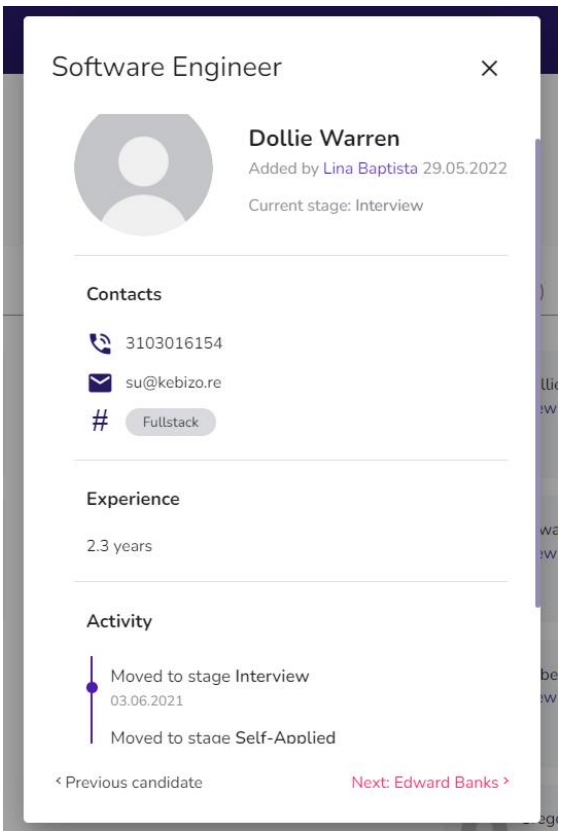


Рисунок 5.8.3 Інформація аппліканта

5.9 Сторінка Interviews

Відображає всі заплановані та автоматично створені інтервью, а також рекрутер отримую сповіщення на пошту за під години до початку відео дзвінка (Рис. 5.9.1).

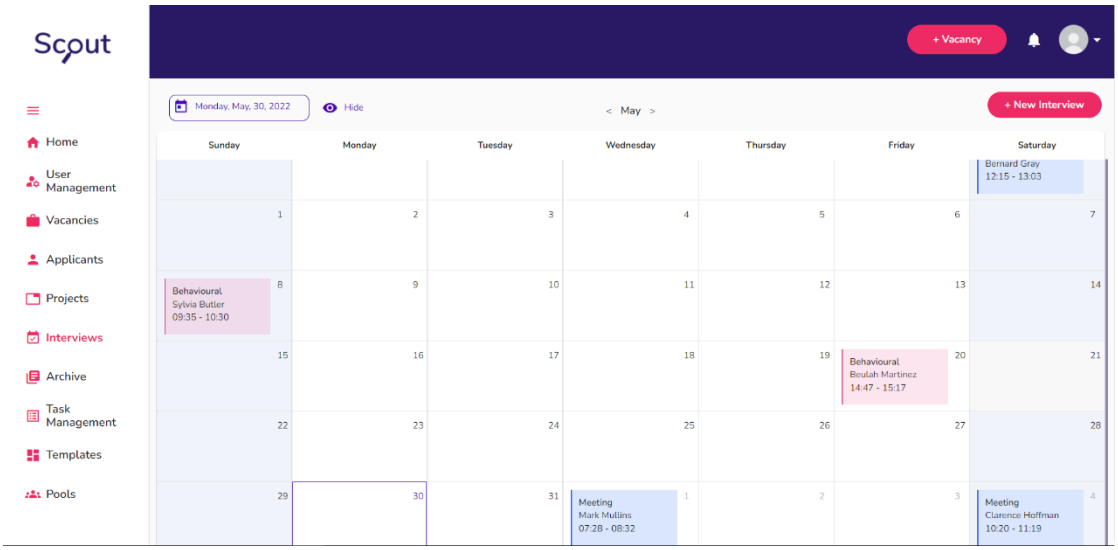


Рисунок 5.9.1 Сторінка Interviews

5.10 Сторінка Task Management

Сторінка для планування задач рекрутера. До вкладки To-do відносяться активні. Done відображає все виконані. А на вкладці Need Review знаходяться автоматично створенні (Рис. 5.10.1).

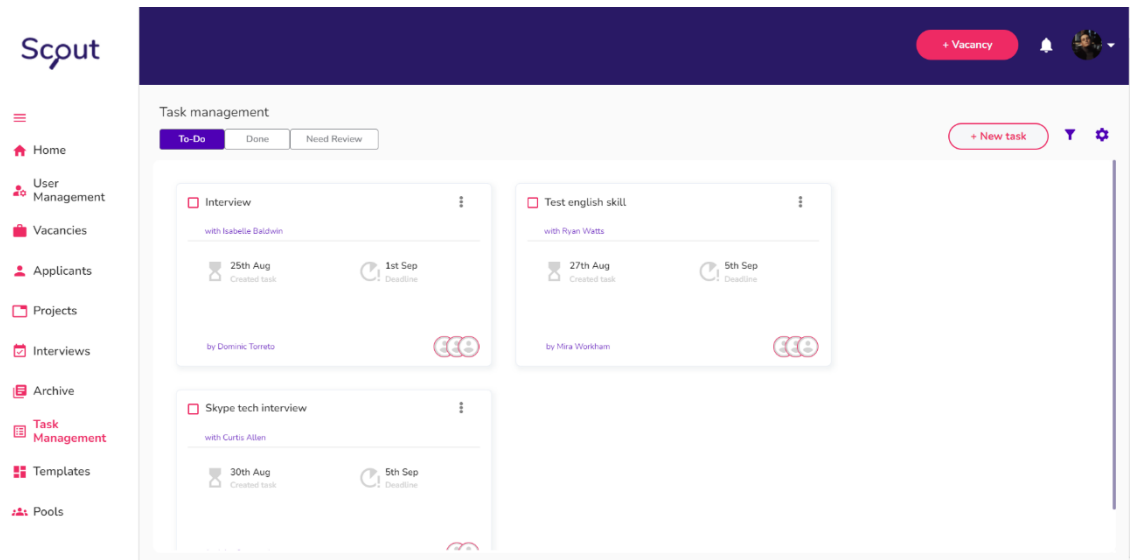


Рисунок 5.10.1 Сторінка Task Management

5.11 Сторінка Templates

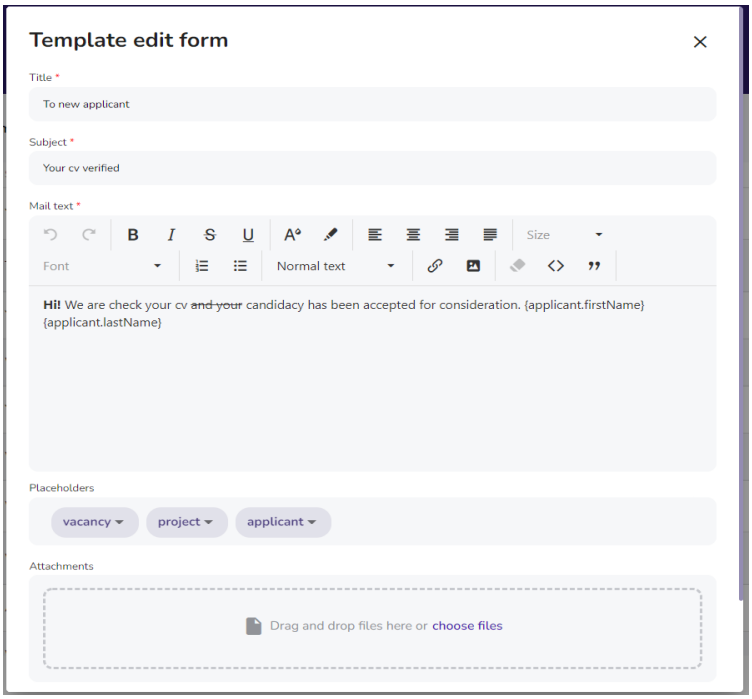


Рисунок 5.11.1 Редагування шаблону

Сторінка для створення та редагування шаблонів повідомлень. Є можливість форматування тексту аналогічно до вордівського, також є плейсхолдери які автоматично будуть замінені при надсиланні листа, або ж можуть буди вибранні при мануальному надсиланні листа (Рис. 5.11.1).

5.12 Сторінка Pools

Для створення груп кандидатів і пришвидшення їх подання на кілька вакансій (Рис. 5.12.1).

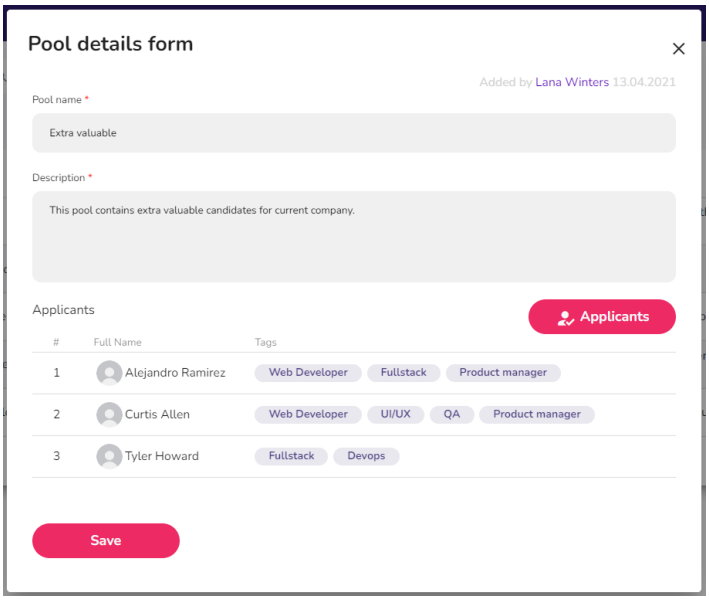


Рисунок 5.12.1 Редагування пулу

РОЗДІЛ 6. ВИКОРИСТАННЯ МАШИННОГО НАВЧАННЯ ТА ХМАРНИХ ТЕХНОЛОГІЙ ДЛЯ ПАРСИНГУ ПДФ

Для парсингу файлів я використав сервіси хмарних технологій надані Амазоном. Для цього потрібно налаштувати AWS Console. Створити аккаунт, та додати API ключі до змінних середовища [18] (Рис. 6.1, 6.2, 6.3).

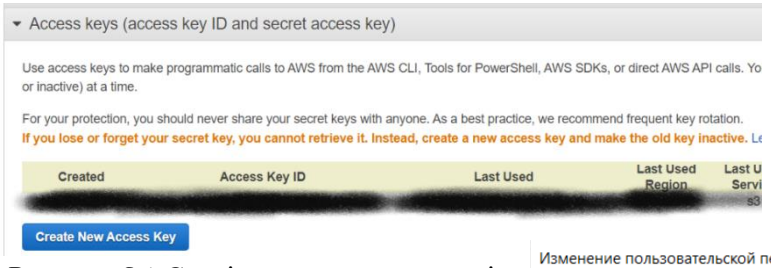


Рисунок 6.1 Сторінка додавання ключів

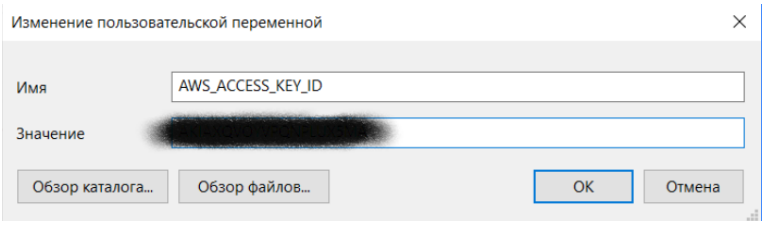


Рисунок 6.2 Додавання ID ключа

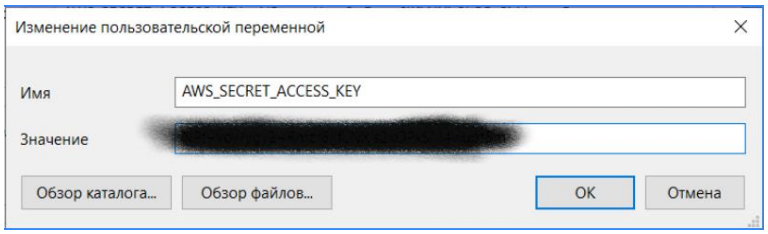


Рисунок 6.3 Додавання ключа

Надалі нам потрібно створити та налаштувати окремо, а потім зв’язати в один флоу (див. Рисунок 6.4 і 6.5) усі сервіси на AWS, що нам потрібні. У наступних підрозділах буде описане налаштування.

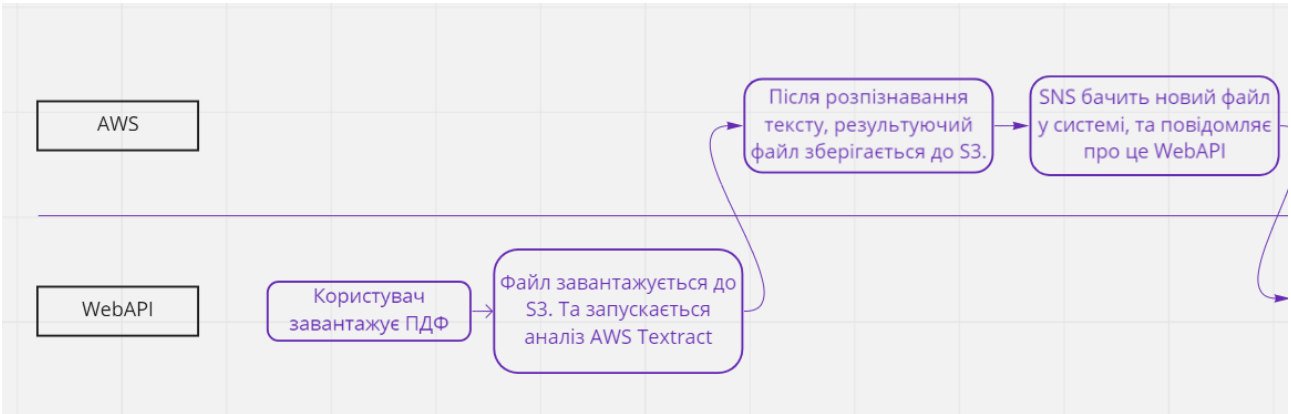


Рисунок 6.4 Флоу виконання парсингу файлу. Частина 1

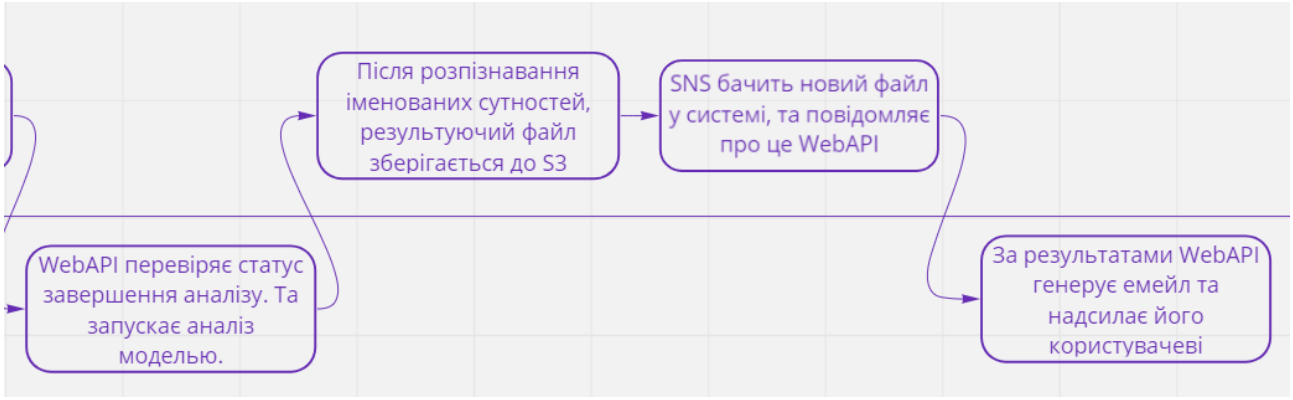


Рисунок 6.5 Флоу виконання парсингу файлу. Частина 2

6.1 S3 bucket

Для зберігання пдф файлів та результатів було створено публічний бакет (Рис. 6.1.1, 6.1.2).

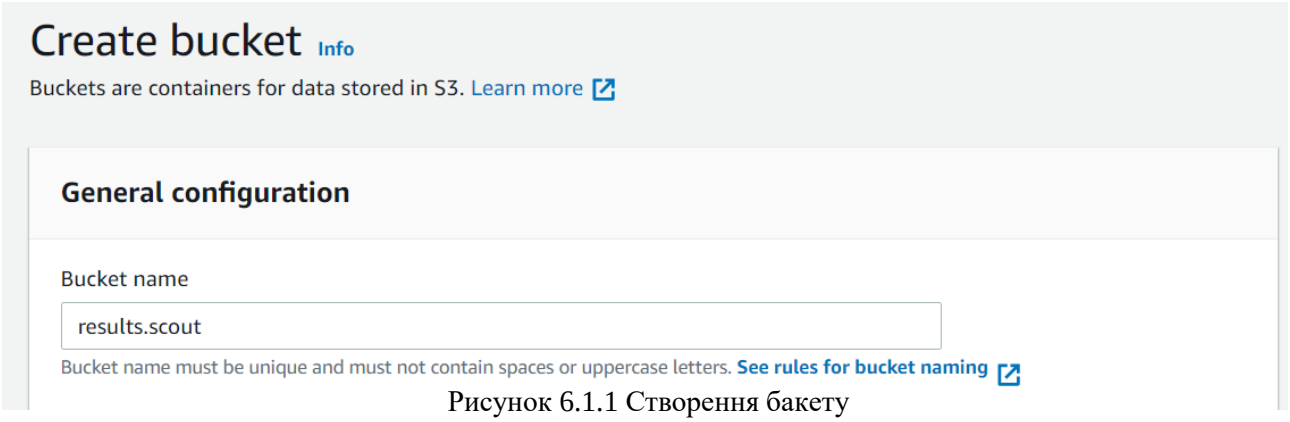


Рисунок 6.1.1 Створення бакету

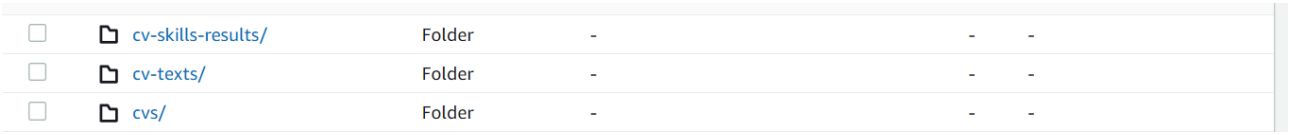
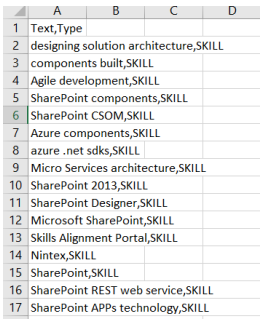


Рисунок 6.1.2 Папки у бакеті

6.2 AWS Comprehend та навчання моделі



	A	B	C	D
1	Text,Type			
2	designing solution architecture,SKILL			
3	components built,SKILL			
4	Agile development,SKILL			
5	SharePoint components,SKILL			
6	SharePoint CSOM,SKILL			
7	Azure components,SKILL			
8	azure .net sdks,SKILL			
9	Micro Services architecture,SKILL			
10	SharePoint 2013,SKILL			
11	SharePoint Designer,SKILL			
12	Microsoft SharePoint,SKILL			
13	Skills Alignment Portal,SKILL			
14	Nintex,SKILL			
15	SharePoint,SKILL			
16	SharePoint REST web service,SKILL			
17	SharePoint APPs technology,SKILL			

Рисунок 6.2.1 Приклад датасету

Для навчання моделі на базі сервісів AWS були створені датасет для навчання та тестовий. Датасети представляють собою список прикладів різних іменованих сутностей які можуть бути в резюме людини (Рис. 6.2.1). Надалі можливе перенавчання моделі при збільшенні датасетів. Два датасети було завантажено до S3 бакету, для того, щоб потім мати до них доступ при навчанні.

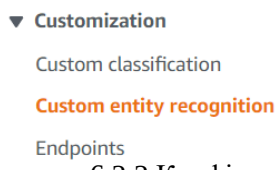


Рисунок 6.2.2 Конфігурація

Далі було налаштовано та навчено модель AWS Comprehend. Обрано була модель для навчання Custom Enitity recognition [19] (Рис. 6.2.2). Створюємо нову модель, та надаємо датасети для навчання (Рис. 6.2.3). Після чого наша модель буде навчатися і як результат ми отримаємо функціонуючу модель для розпізнавання інформації з тексту резюме (Рис. 6.2.4).

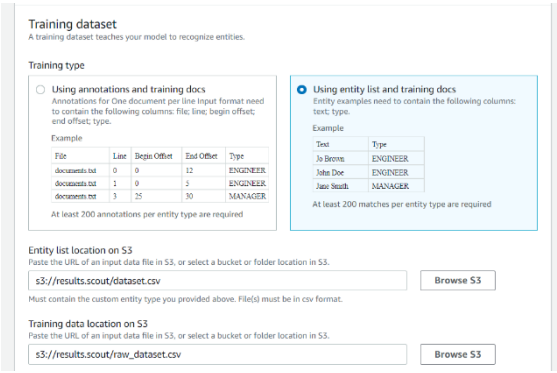


Рисунок 6.2.3 Датасети для навчання

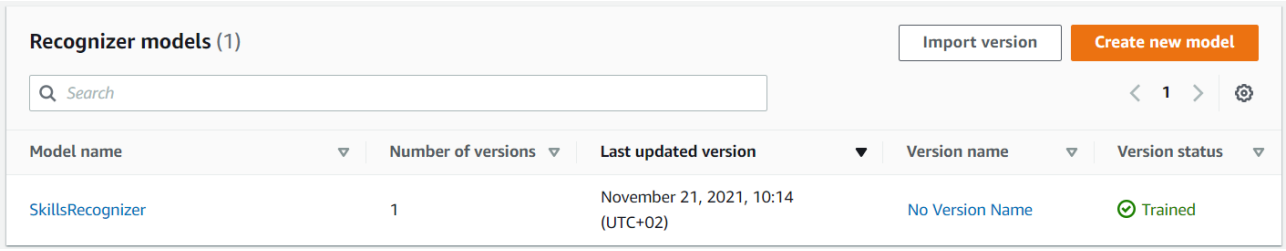


Рисунок 6.2.4 навченна модель

6.3 AWS Textract

Textract – це вже навчена модель для розпізнавання тексту яку ми можемо використати без додаткового ванчання. Створюючи сутність інтерфейсу IAmazonTextract [20] DI забезбечує нас сутністю за допомогою якої можливо зробити аналіз і потім використавши IS3Uploader [20] завантажити файли до S3 (Рис. 6.3.1) .

```
GetDocumentTextDetectionResponse response = await _textract.GetDocumentTextDetectionAsync(request);
```

Рисунок 6.3.1 Використання Textract в коді

ВИСНОВОК

У роботі розроблена системи (клієнт-сервіс) для автоматизації процесів найму, відбору та централізації інформації про працівників з використанням машинного навчання та хмарних технологій з найкращим функціоналом для додавання працівників.

Було розглянуте питання побудови надійної архітектури та написання якісних сервісів та рівнів. Запропоновано основні положення, ідеї та підходи для створення надійних та підтримуваних додатків (frontend та backend).

В процесі виконання роботи було розібрано ключові етапи проектування додатку та проаналізовано найвигідніші стратегії.

Отримано такі основні результати:

1. Спроектовано та розглянуто структуру onion архітектури.
2. Спроектовано та побудовано структуру модульно-компонентної архітектури.
3. Створений повноцінний додаток, який буде впроваджено в життя.
4. Реалізована та навчена модель для розпізнавання вмінь з резюме.
5. Було реалізовано розділення по ролям, авторизацію зі збереженням та автовідновленням сесії.
6. Було створене google chrome розширення для парсингу Linkedin.

Отже, розроблений якісний сервіс для автоматизації процесів найму, який можливо у подальшому модифікувати, розширювати.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Офіційний сайт проекту Material Design. Ел. ресурс. Режим доступу: https://material.io/design.
2. Мартін Р.С. Чиста архітектура. – Вид-во: Фабула, 2019. – 368с.
3. Angular. The modern web developer's platform. Ел. ресурс. Режим доступу: https://angular.io/guide/releases#support-policy-and-schedule.
4. Angular Material. Material Design components for Angular. Ел. ресурс. Режим доступу: https://material.angular.io/.
5. The Onion Architecture. Jeffrey Palermo. Ел. ресурс. Режим доступу: https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/
6. Офіційний сайт платформи ASP.NET. Ел. ресурс. Режим доступу: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet
7. Fowler M. Inversion of Control Containers and the Dependency Injection pattern, 2006. – Ел. ресурс. Режим доступу: http://martinfowler.com/articles/injection.html.
8. CQRS (command query responsibility segregation). Matt Heusser. Ел. ресурс. Режим доступу: https://www.techtarget.com/searchapparchitecture/definition/CQRS-command-query-responsibility-segregation
9. Use MediatR in ASP.NET or ASP.NET Core. Ashish Patel. Ел. ресурс. Режим доступу: https://medium.com/dotnet-hub/use-mediatr-in-asp-net-or-asp-net-core-cqrs-and-mediator-in-dotnet-how-to-use-mediatr-cqrs-aspnetcore-5076e2f2880c
10. Офіційний сайт Microsoft LINQ. Ел. ресурс. Режим доступу: https://docs.microsoft.com/ru-ru/dotnet/csharp/linq/
11. Using Dapper with ASP.NET Core Web API. Marinko Spasojevic. Ел. ресурс. Режим доступу: https://code-maze.com/using-dapper-with-asp-net-core-web-api/

12. Офіційний сайт JSON Web Token. Ел. ресурс. Режим доступу: https://jwt.io/introduction
13. Офіційний сайт Elasticsearch. Ел. ресурс. Режим доступу: https://www.elastic.co/
14. Офіційний сайт Amazon Textract. Ел. ресурс. Режим доступу: https://aws.amazon.com/textract/
15. Офіційний сайт Amazon Comprehend. Ел. ресурс. Режим доступу: https://aws.amazon.com/comprehend/
16. David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. – Вид-во: Lingvisticæ Investigationes, 2017. – 20с..
17. Офіційний сайт Angular. Ел. ресурс. Режим доступу: https://www.typescriptlang.org/docs/handbook/angular.html
18. AWS API Integration. Ел. ресурс. Режим доступу: https://support.zeustrack.io/support/solutions/articles/35000113493-aws-api-integration
19. Build a custom entity recognizer using Amazon Comprehend. Ел. ресурс. Режим доступу: https://aws.amazon.com/ru/blogs/machine-learning/build-a-custom-entity-recognizer-using-amazon-comprehend/
20. Офіційний сайт пакету AWSSDK.Core. Ел. ресурс. Режим доступу: https://www.nuget.org/packages/AWSSDK.Core/