**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА**
Факультет комп'ютерних наук та кібернетики
Кафедра та теорії та технології програмування

# Кваліфікаційна робота
**на здобуття ступеня бакалавра**

за спеціальністю 122 Комп'ютерні науки

На тему:

## ВЕБЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ ПРОЦЕСУ ПРАЦЕВЛАШТУВАННЯ

Виконала студенка 4-го курсу
Софія БІЛІНСЬКА

(підпис)

Науковий керівник:
Доцент, кандидат фіз.-мат. наук
Людмила ОМЕЛЬЧУК

(підпис)

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

Студент

(підпис)

Роботу розглянуто й допущено до захистуна засіданні кафедри теорії та технологій програмування
«_5_»___червня_____2023 р.,
протокол № _18_
Завідувач кафедри
Микола НІКІТЧЕНКО

(підпис)

Київ-2023

# РЕФЕРАТ

Обсяг роботи 55 сторінки, 39 ілюстрацій, 15 таблиць, 22 джерел посилань, 1 додаток.

ASP.NET, ANGULAR, TYPESCRIPT, ENTITYFRAMEWORK, WEBAPI, CQRS, ВЕБЗАСТОСУНОК

Об’єктом роботи є процес пошуку роботи кандидатами та найму співробітників в компанії. Предметом роботи є вебзастосунок для організації процесу пошуку роботи та автоматизації роботи агенції з працевлаштування.

Метою кваліфікаційної роботи є проєктування та розробка вебзастосунку «BoardingTracker», який є системою для рекрутерів та кандидатів і дає змогу користувачам застосунку переглядати доступні вакансії та відгукуватись на них. А рекрутерам додавати нові вакансії та організовувати зустрічі з кандидатами.

Методи розробки: в якості базового інструменту створення даного додатку було використано ASP.NET Core Web API (використовується для серверної частини) та Angular (використовується для клієнтської частини), для бази даних було використано СУБД SQL Server. В якості інтегрованих середовищ розробки було використано Visual Studio 2022 та Visual Studio Code.

Результати роботи: досліджено основні етапи розробки вебзастосунку та досліджено різні архітектурні підходи для проєктування програмного забезпечення і в результаті дослідження використано підхід CQRS. Розроблено та протестовано вебзастосунок «BoardingTracker» для організації процесу працевлаштування.

Розроблений програмний засіб може бути корисним для роботодавців, які шукають нових робітників для своєї компанії, а також для людей, які шукають нову роботу відповідно до своїх кваліфікацій та вимог.

**ЗМІСТ**

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

**ADO.NET** – ActiveX Data Objects .NET (об’єкти даних ActiveX .NET);

**API** – Application Programming Interface (прикладний програмний інтерфейс);

**ASP.NET** – Active Server Pages .NET (активні серверні сторінки .NET);

**AWS** – Amazon Web Services (веб сервіси Amazon)

**CQRS** – command-query responsibility segregation (принцип розмежування відповідальності команд-запитів);

**CRUD –** Create, Read, Update and Delete (операції створення, читання, оновлення та видалення даних);

**EF Core** – Entity Framework Core (фреймворк сутностей);

**HR** – Human resources (людські ресурси);

**HTTP** – HyperText Transfer Protocol (протокол передачі гіпертексту);

**JSON** – JavaScript Object Notation (текстовий формат обміну даними);

**JWT** – JSON Web Token (веб-токен JSON);

**MS SQL Server** – Microsoft SQL Server (Microsoft SQL сервер);

**REST** – Representational State Transfer (передача репрезентативного стану);

**SQL** – Structured query language (мова структурованих запитів);

**ООП** – об’єктно-орієнтоване програмування;

**ПЗ** – програмне забезпечення;

**СУБД** – система управління базами даних.

# ВСТУП

**Оцінка сучасного стану об’єкта розробки.** У сучасному світі цифрові технології відіграють важливу роль, оскільки бізнес-сфера активно використовує цифровізацію. Цифрові компетентності як вимогу для моніторингу якості персоналу описують В. Куйбіда, О. Петроє, Г. Андрощук, Г. Федулова [1]. Цифрові технології менеджменту людських ресурсів досліджують Н. Нагибіна та А. Щукина [2] та С. Рудакова, Н. Данилевич, Л. Щетініна [3], управління та розвиток інтелектуального потенціалу підприємства обґрунтовує Й. Ситник [4], а використання моделі корпоративних компетенцій для різних категорій персоналу з метою підвищення ефективності його діяльності – Г. Захарчин [5]. Цифровізація кадрової політики регламентована Кодексом законів про працю України, підтримується державним додатком “Дія”. Найм працівників регламентують також накази Державного комітету статистики України. Це розкрито у монографії [6] та дає змогу істотно підвищити рівень вимог до компетентностей на ринку праці. Дані дослідження були наведені в статті [7].

Завдяки розвитку цифрових інструментів, працедавці та працівники мають доступ до багатьох нових можливостей для пошуку та найму робочої сили. Веб-додатки дозволяють розміщувати вакансії та шукати кандидатів на роботу в Інтернеті.

**Актуальність роботи та підстави для її виконання.** З розвитком технологій з'явилась потреба в створенні вебзастосунків, які б спрощували рутинні завдання та ефективно організовували процес працевлаштування. Автоматизація процесу працевлаштування це:

- можливість витрачати менше часу на обробку даних;
- цифровізація першого етапу проведення співбесіди з кандидатами;

- формування єдиної бази даних вакансій та кандидатів;
- пришвидшення процесу найму та пошуку нових працівників;
- покращення ефективності роботи HR-відділів.

Таким чином, задача розробки вебзастосунку для організації процесу працевлаштування є актуальною задачею.

**Мета й завдання роботи.** Метою кваліфікаційної роботи є проєктування та розробка вебзастосунку «BoardingTracker», який є допоміжним засобом для HR-менеджерів, рекрутерів та кандидатів і дає змогу кандидатам переглядати доступні вакансії та відгукуватись на них. А рекрутерам додавати нові вакансії та організовувати зустрічі з кандидатами. Застосування розробленого застосунку сприятиме підвищенню ефективності роботи HR-менеджерів, рекрутерів компаній. Для досягнення цієї мети поставлено такі завдання:

- визначити ключові функціональні вимоги до вебзастосунку;
- дослідити наявні технології для розробки та архітектурні підходи;
- розробити вебзастосунок на основі вимог та за допомогою вибраних технологій та створеної архітектури;
- протестувати розроблений додаток;
- поглибити та закріпити навички роботи з обраними технологіями.

**Об'єкт, методи й засоби розроблення.** Об’єктом роботи є процес пошуку роботи кандидатами та найму співробітників в компанії. Предметом роботи є вебзастосунок для організації процесу пошуку роботи та автоматизації роботи HR-менеджерів компаній та агенції з працевлаштування.

Розробка вебзастосунку базувалась на таких принципах. Спочатку була повністю розроблена база даних для її проєктування було використано СУБД SQL Server та серверна частина програми з використанням ASP.NET Web.API. У розробці backend було використано Onion архітектуру і дотримано всі вимоги RESTful API. Також для забезпечення кращої масштабованості додатку був використаний архітектурний підхід для проєктування

програмного забезпечення – CQRS. Потім на основі API запитів було реалізовано клієнтську частину з використанням Angular.

В якості інтегрованих середовищ розробки було використано Visual Studio 2022 та Visual Studio Code.

# РОЗДІЛ 1. ОГЛЯД НАЯВНИХ НА РИНКУ ЗАСТОСУНКІВ

На сьогоднішній день існує велика кількість вебзастосунків для автоматизування пошуку роботи. Існують локальні системи, які використовуються всередині компаній та глобальні, які орієнтовані на різні сфери і на велику кількість користувачів. Розглянемо деякі з них.

Система **PeopleForce** [18] – це система автоматизації підбору персоналу, яка дозволяє спростити процес найму. Дана система орієнтована на внутрішнє використання в компанії для пошуку відповідних кандидатів. Додаток дозволяє формувати базу даних кандидатів з їхніми резюме та проводити їх скрінінг. Система надає аналітичні засоби для відстеження та оцінки ефективності рекрутингових процесів. Інтерфейс програми показано на Рисунок 1.

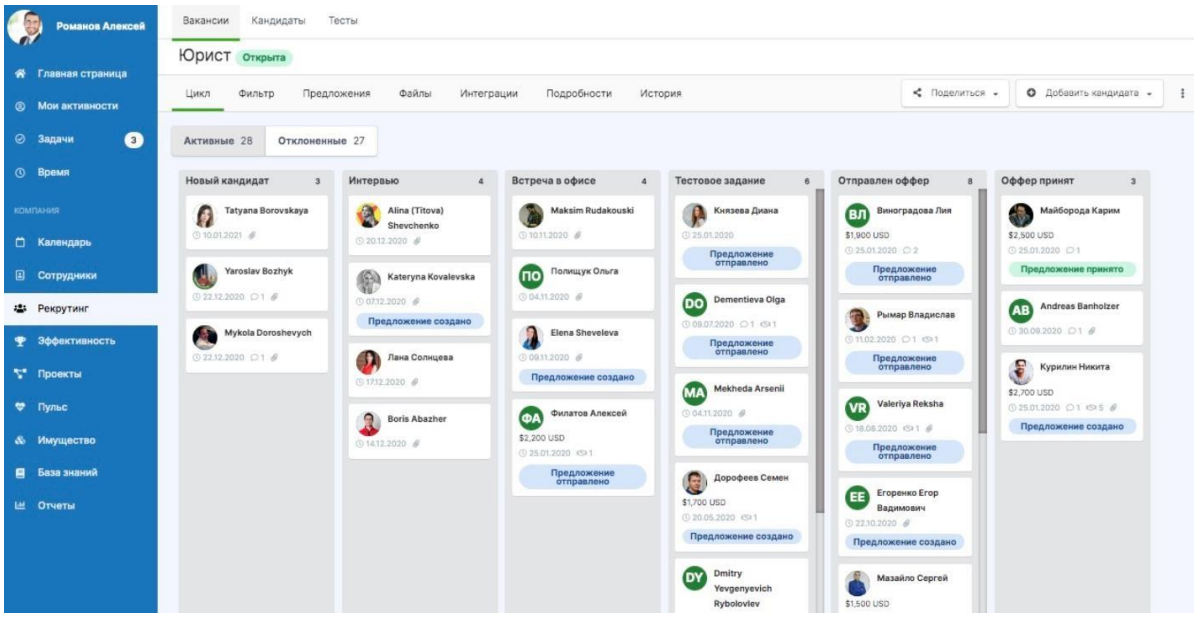


**Рисунок 1** – Інтерфейс системи PeopleForce

З недоліків даної системи, що вона орієнтована на компанії з великим штатом працівників, для менших компаній з обмеженим бюджетом, використання програми буде зв’язане зі значними витратами. Також до недоліків можна віднести – складність впровадження. Впровадження нової

системи в компанію може вимагати часу та зусиль для навчання персоналу та впровадження необхідних процесів.

Система **Coloris HRM** [19] – це програмне забезпечення для управління людськими ресурсами, яке орієнтоване на внутрішне використаннє в компаніях, яке допомагає автоматизувати різні аспекти управління персоналом. Система містить в собі розширені інструменти звітності та аналізу, які охоплюють всі аспекти управління персоналом: від онбордингу та оцінки до навчання, управління талантами та розвитку кар'єри. Тож додаток має широкий функціонал, до недоліків можна віднести вартість, потрібен час на впровадження в компанію. Інтерфейс програми показано Рисунок 2.

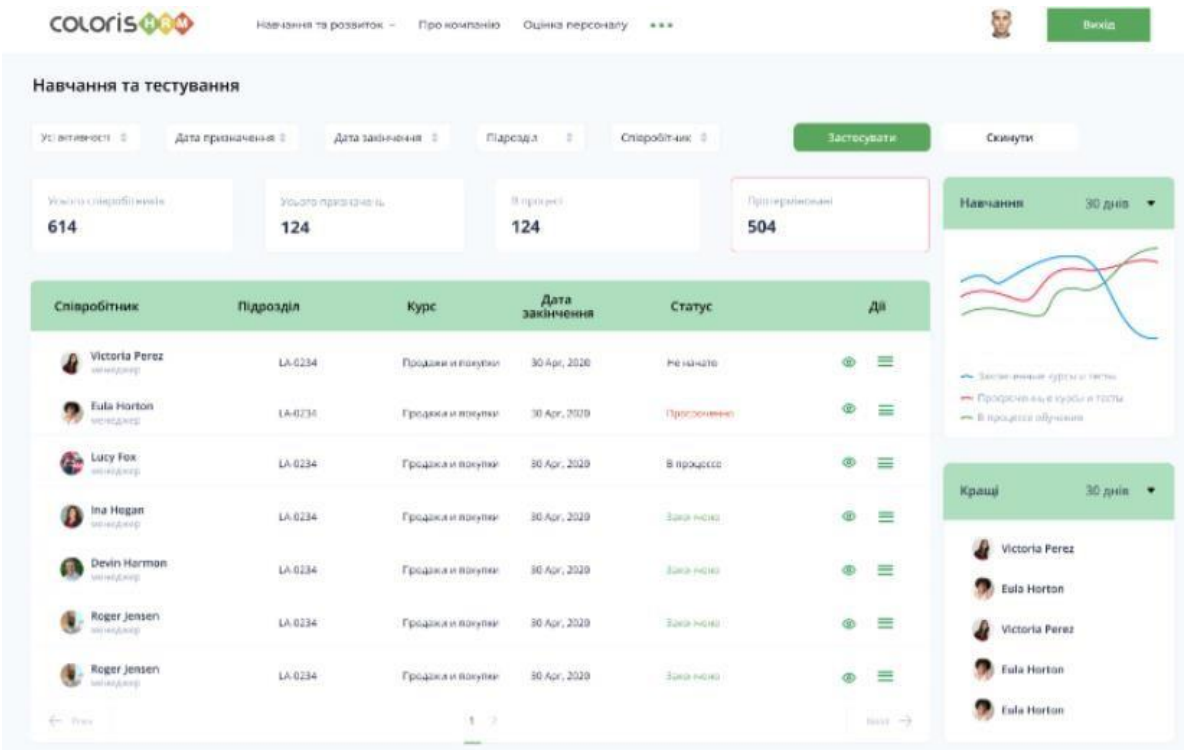


**Рисунок 2** – Інтерфейс Caloris HRM

Портал **Djinni** [20] – це рекрутинговий портал та платформа для пошуку роботи. Дана система відноситься до глобальних так як орієнтована на велику кількість користувачів з різними напрямами. Вебзастосунок пропонує різні інструменти для пошуку роботи, різні види фільтрів, що допомагає кандидатам знайти вакансії, які найкраще відповідають їхнім потребам та

навичкам. Для рекрутерів Djinni надає зручний інтерфейс для управління вакансіями та кандидатами. Інтерфейс програми показано на Рисунок 3. Рекрутери можуть вести активну співпрацю з кандидатами через платформу Djinni. Вони можуть надсилати повідомлення, проводити співбесіди в онлайн-режимі, отримувати оновлення про статус кандидатів. До недоліків можна віднести обмежену функціональність, так як що використовувати внутрішню HR-систему вона буде містити в собі більший функціонал.

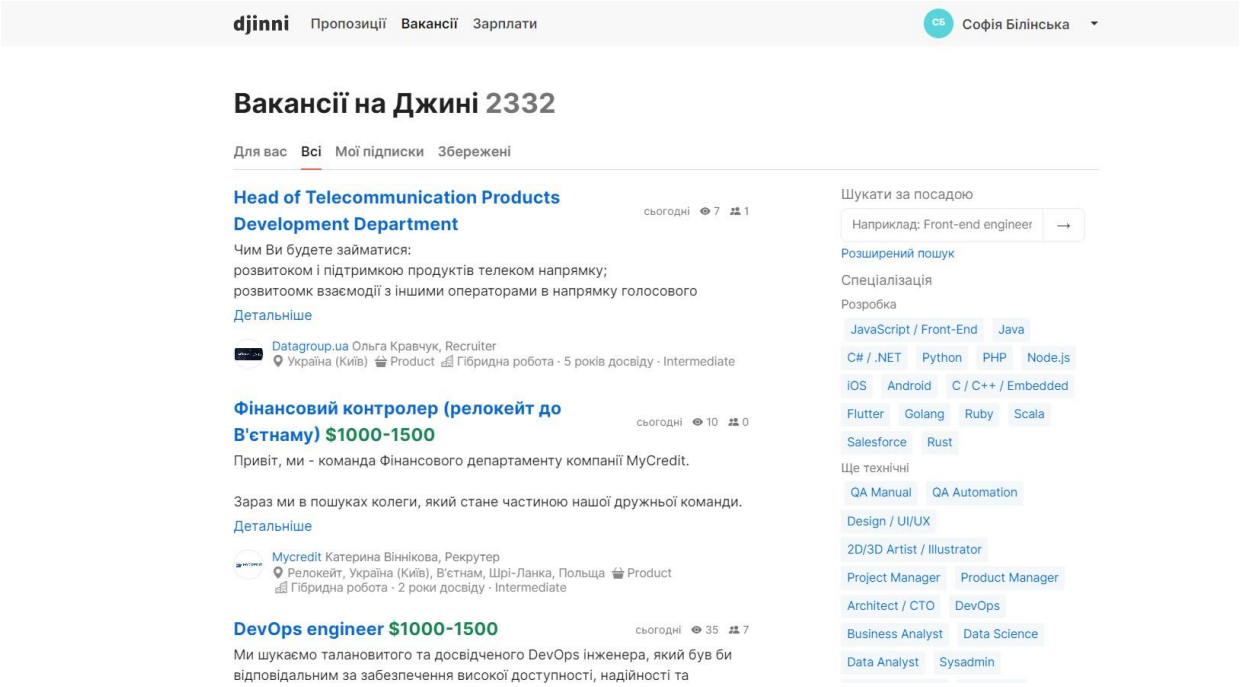


**Рисунок 3** – Інтерфейс Djinii

До глобальних сервісів можна віднести **LinkedIn** [21] – це платформа для спілкування рекрутерів та кандидатів. LinkedIn є одним з найбільш популярних професійних мережевих ресурсів, де користувачі можуть створювати свої профілі, розміщувати резюме та спілкуватися з потенційними роботодавцями. Рекрутери також можуть активно використовувати LinkedIn для пошуку та зв'язку зі спеціалістами у своїй галузі. Платформа надає різноманітні інструменти для пошуку кандидатів, зокрема фільтри по галузі, навичках, регіоні та досвіду роботи. Кандидати також можуть активно

відстежувати вакансії та компанії, які їх цікавлять, і надсилати свої заявки на вакансії безпосередньо через платформу (див. Рисунок 4). Один з головних переваг LinkedIn полягає у його глобальному охопленні. Це дозволяє рекрутерам та кандидатам знаходити можливості та взаємодіяти зі спеціалістами з різних країн і регіонів. До недоліків цієї мережі можна віднести його комерційну природу. Багато зручних функцій та інструментів доступні за плату або з обмеженими можливостями у безкоштовній версії.

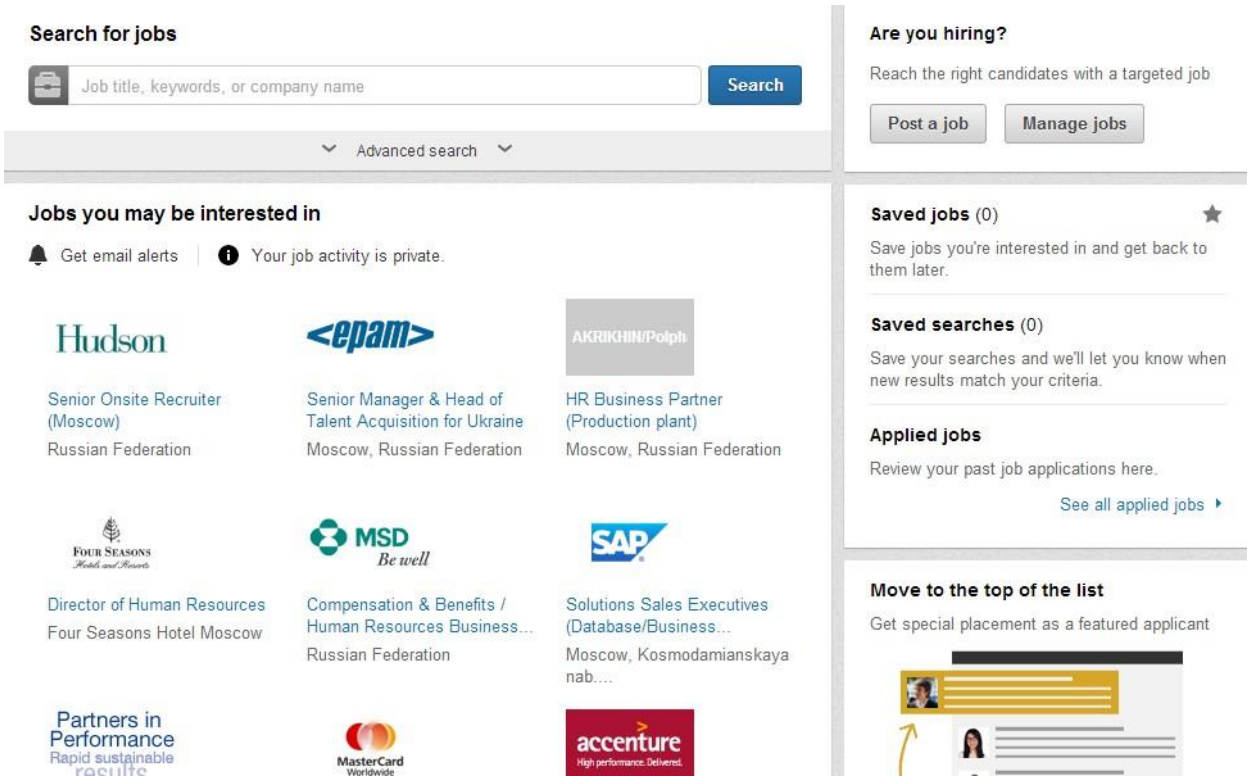


**Рисунок 4** – Інтерфейс LinkedIn

Здійснений аналіз ринку систем автоматизації підбору персоналу показав, що існує різноманітність інструментів та платформ, які допомагають компаніям спростити та оптимізувати процес найму. Кожна з розглянутих систем – PeopleForce, Coloris HRM, Djinni, LinkedIn – має свої переваги та недоліки, але вони спрямовані на різні сегменти ринку.

Отже, розробка вебзастосунку для автоматизації працевлаштування є актуальною, оскільки вона допомагає компаніям зекономити час, зусилля та

ресурси, покращує якість відбору кандидатів і сприяє швидкому та ефективному процесу найму. Такий вебзастосунок може мати різноманітні функціональні можливості, які спрямовані на полегшення роботи рекрутерів і поліпшення досвіду користувачів.

# РОЗДІЛ 2. ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

Розглянемо інструменти, технології та архітектурні рішення, які дозволять виконати раніше поставлені цілі найефективнішим способом. Однією з цілей даної роботи було ознайомитися з новими архітектурними засобами та технологіями і в результаті навчання розробити вебзастосунок.

## 2.1. Вибір технологій серверної частини

Найбільш важливою частиною розробки вебзастосунку є проєктування архітектури, так як правильно спроєктована архітектура стає більш гнучкою та спрощує подальшу підтримку застосунку. При реалізації застосунку було використано Onion архітектуру.

***Onion архітектура*** [8] – це архітектурний підхід до розробки програмного забезпечення, заснований на принципах SOLID [9] та інших принципах ООП. Основна ідея архітектури Onion полягає в тому, що програмне забезпечення має бути розділене на логічні рівні, які взаємодіють один з одним через інтерфейси. Кожен рівень має власну реакцію та відповідальність за певну частину функціональності системи (див. Рисунок 5).

Основні рівні Onion архітектури, та додаткові рівні, які були використані:

- Domain layer (доменний рівень) – це центральний рівень, який містить бізнес-логіку системи. Цей рівень не залежить від жодного іншого.
- Application layer (рівень застосування) – цей рівень містить логіку застосування та взаємодіє з іншими рівнями. Взаємодіє з бізнес-логікою за допомогою інтерфейсів рівня домену.
- Infrastructure layer (інфраструктурний рівень) – цей рівень містить реалізацію інтерфейсів з доменного та рівня застосування. Він

містить також зовнішні залежності системи, такі як бази даних, мережеві служби та інші системи.

- Persistance layer (рівень збереження) – це додатковий рівень, який можна додати до Onion архітектури. Його основна відповідність – збереження та витягування даних з бази даних.
- Presentation layer (рівень представлення) – це рівень Onion архітектури, який відповідає за представлення інформації користувачеві та обробку дій користувача.
- Test layer (рівень тестування) – це додатковий рівень, який можна додати до Onion архітектури, і він відповідає за тестування системи та її компонентів.

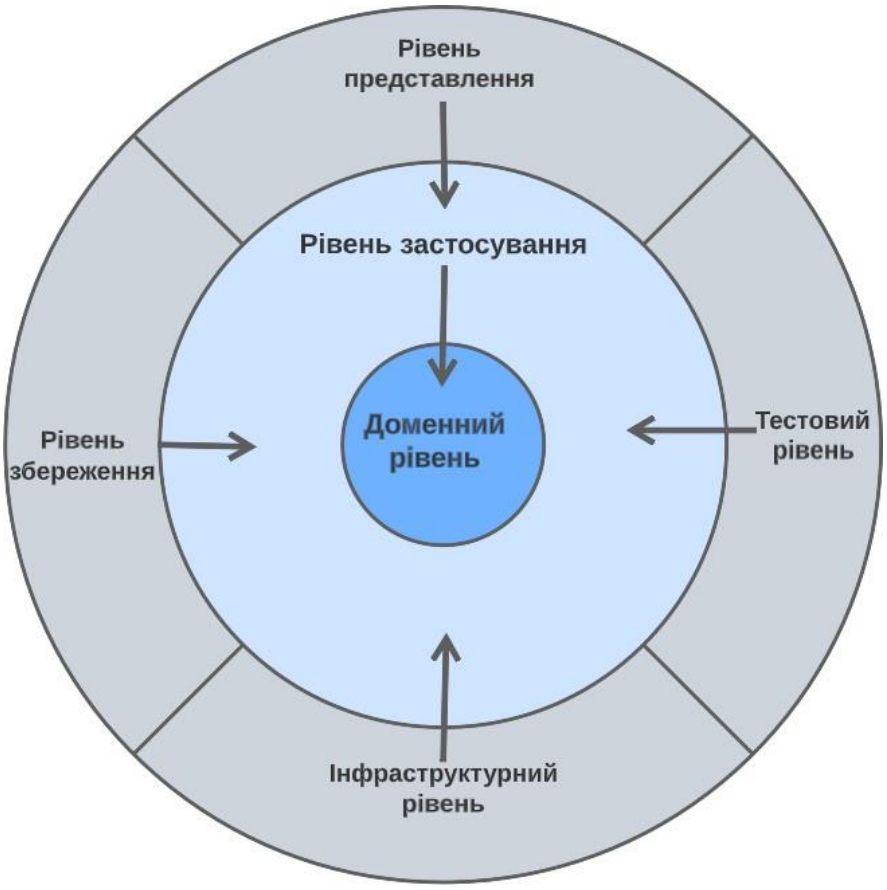


**Рисунок 5** – Onion архітектура

Для реалізації backend частини проєкту було обрану технологію ***ASP.NET Web API***, так як цей фреймворк дозволяє легко створювати RESTful

вебсервіси, які можуть бути доступні через HTTP-протокол. Дана технологій дозволить зручно здійснювати обмін даними між різними системами, що в подальшому дозволить нам з'єднати HTTP-методи з фронтенд частиною додатку. Відповідно до обраної технологій мовою програмування являється C#.

***Entity Framework Core*** [10] – це комплекс інструментів в ADO.NET, які підтримують розробку програм, які орієнтовані на дані. При розробці програмного забезпечення було використано підхід Database First і дана технологія дозволила автоматично створити моделі на базі створеної бази даних. Див. додаток А.

У розробленому проєкті моделі даних могли змінюватись по мірі реалізації функціоналу. При додаванні або зміні нових сутностей або властивостей схеми бази даних повинні бути відповідним чином змінені для синхронізації з додатком. Для цього було використано функцію **міграції в Entity Framework Core**, яка дає змогу послідовно застосовувати зміни схеми до бази даних, щоб синхронізувати її з моделлю даних у застосунку без втрати наявних даних. Застосування міграцій наведено на Рисунок 6.

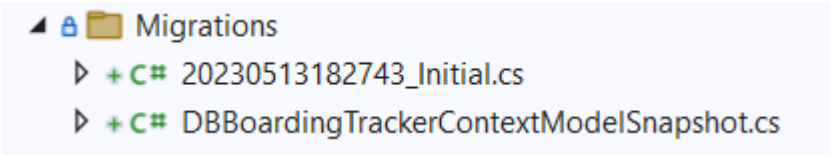


**Рисунок 6 –** Застосування міграцій

В якості системи для управління базами даних було використано MS SQL Server, так як вона чудово підтримується EF Core та є зручною та багатофункціональною у використанні.

Як згадувалось раніше, однією з цілей було дослідження нових архітектурних підходів. ***CQRS*** [11] – це шаблон, який розділяє операції читання і запису на різні моделі, використовуючи команди для оновлення даних і запити для читання даних.

Команда - це інструкція для виконання конкретного завдання. Це намір щось змінити. Команда виконує роботу, але не повертає результату. Запит – це запит на інформацію. Запит повертає результат, але не змінює стан. Ілюстрація з роботою CQRS шаблону наведена на Рисунок 7.

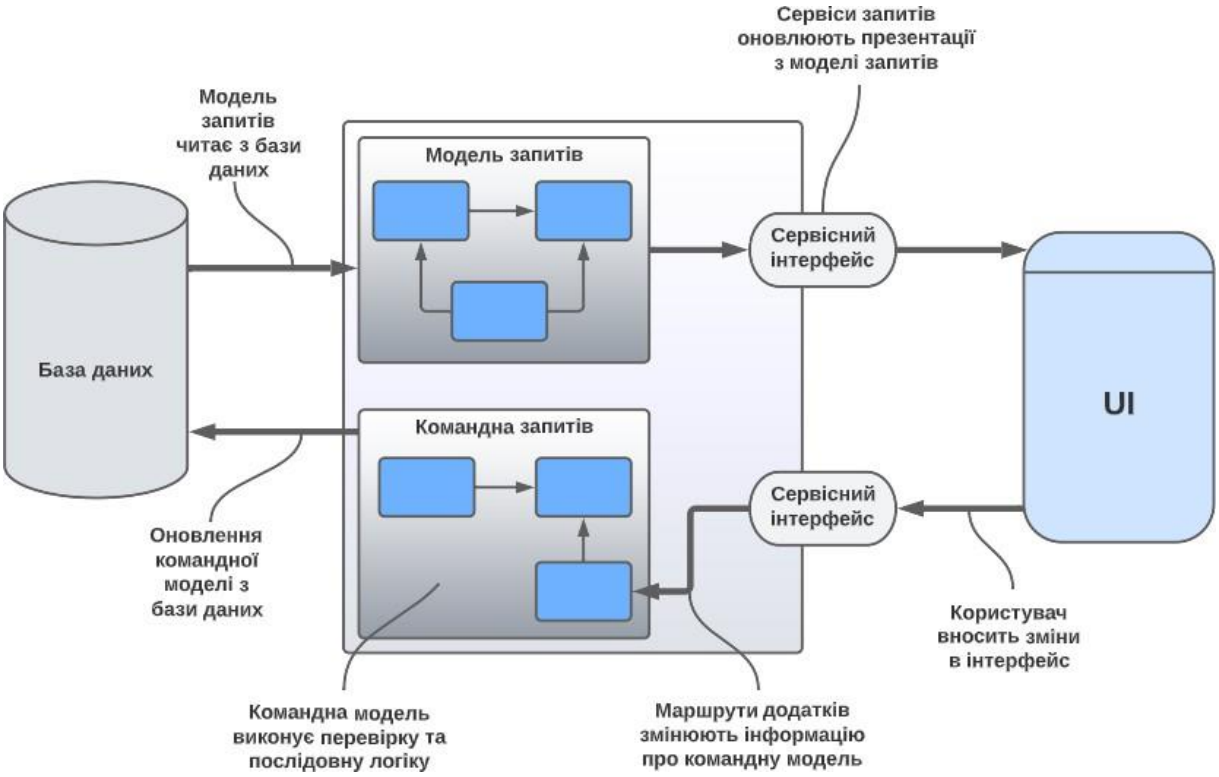


**Рисунок 7** –Логіка роботи CQRS паттерну

До переваг CQRS належать:

- Чіткі кордони між поведінкою системи. Він містить конкретні рекомендації щодо побудови програм, заснованих на поведінці.
- Слабкий зв'язок. Логіка зв'язана і менш взаємопов'язана, що спрощує створення модульних додатків, які можна підтримувати.
- Зниження когнітивного навантаження. Вертикальний поділ зберігає пов'язаний код зберігається разом. Що дозволяє легше зосередитися на конкретному завданні.

- Незалежне масштабування. CQRS дозволяє незалежно масштабувати робочі навантаження читання та запису і може призвести до меншої кількості конфліктів блокувань.
- Передбачуваність. Оскільки поділ окремий, зберігання логіки запису та читання окремо зменшує ймовірність спагеті-коду.

Для реалізації даного архітектурного підходу, була використана бібліотека **MediatR**, яка допомагає реалізувати шаблон Посередник (Mediator). Шаблон Посередник просто визначає об'єкт, який інкапсулює спосіб взаємодії об'єктів один з одним. Два або більше об’єктів не залежать безпосередньо один від одного, а взаємодіють за допомогою посередника, який відповідає за передачу їх взаємодії один з одним (див. Рисунок 8).


**Рисунок 8** – Візуалізація роботи шаблону Посередник

Цей шаблон забезпечує "слабкий зв'язок", завдяки цьому, код стає більш простим і легшим для застосування.

**ASP.NET Core Middleware** [12] **–** проміжне програмне забезпечення, яке відповідає за обробку запитів в додатках ASP.NET Core. Проміжне ПЗ створює конвеєр для обробки запитів та відповідей і складається з делегатів, які виконуються один за одним (див. Рисунок 9). Цей механізм забезпечує гнучкий

і розширюваний спосіб опрацювання HTTP-запитів і відповідей, спрощуючи реалізацію складних функцій, як-от автентифікація, обробка помилок.

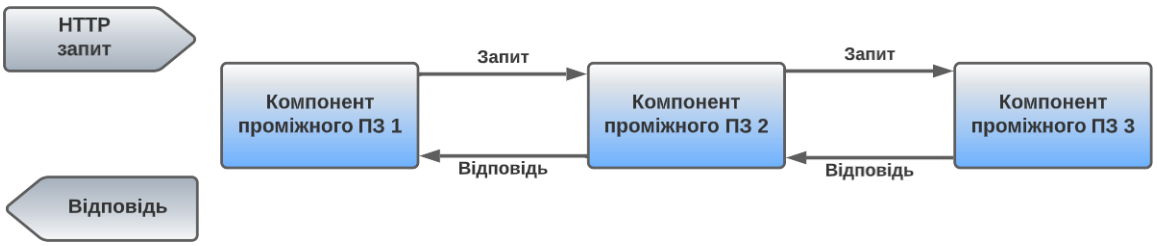


**Рисунок 9** – Візуалізація роботи проміжного ПЗ

Для збереження даних було використано **AWS S3** [13] – що являється хмарним сервісом зберігання даних, наданим компанією Amazon. Він надає безпечне та масштабоване зберігання об'єктів, таких як файли, зображення, відео, аудіо, документи та інші типи даних, у віртуальних контейнерах.

Для авторизації та автентифікації було використано **JSON Web Token (JWT) –** це відкритий стандарт веб-токенів, який використовується, щоб безпечно передавати інформацію у форматі JSON. JWT складається з трьох частин: заголовок (header), корисне навантаження (payload) і підпис або дані шифрування. Кожна частина відокремлена від інших частин крапкою та закодована у форматі Base64. За допомогою JWT токенів авторизація виглядає таким чином:

1) Користувач вводить свої дані авторизації на сервері.
2) Якщо авторизація пройшла успішно, сервер видає клієнту токени доступу та оновлення.
3) Після цього, при кожному запиті до сервера, клієнт надсилає токен доступу, який перевіряє сервер.
4) Якщо токен доступу дійсний, клієнту дозволяється отримати доступ до ресурсів на сервері.

5) Якщо токен доступу закінчився, клієнт може використати токен оновлення для отримання нових токенів доступу та оновлення від сервера.
6) Якщо токен оновлення недійсний, клієнт повинен пройти процес авторизації знову.

## 2.2. Вибір технологій клієнтської частини

Frontend частина вебзастосунку – це частина програмного забезпечення, яка взаємодіє з користувачем через веб браузер. Вона відповідає за те, як користувач буде бачити та взаємодіяти з вебзастосунком. Тому для клієнтської частини було обрано Angular.

**Angular** [15] – це фреймворк з відкритим вихідним кодом, створений компанією Google. В основному дозволяє розробляти односторінкові веб-додатки.

Зараз існує велика кількість різних фреймворків для написання клієнської частини додатку і чистий JavaScript є цілком самодостатнім. В розробці було обрано Angular через низку його переваг:

- Типізація. Для написання коду можна використовувати TypeScript, ця мова є строго типізованою, що зменшує кількість можливих помилок при написанні коду.
- Компонентна архітектура. Це дозволяє краще організувати код, так як кожен компонент містить у інтерфейс та логіку до цього інтерфейсу.
- Модульна архітектура. Дана архітектура дозволяє розділити вебзастосунок на окремі модулі, що робить розробку простішою. Кожен модуль містить в собі компоненти, сервіси і файли, які виконують тестування.

**Angular Material** [16] – це бібліотека компонентів, яка була використана для покращення графічного інтерфейсу.

**SCSS** – це вдосконалена версія CSS, яка допомагає легко створювати та керувати стилями у вебзастосунку. Для розмітки в додатку був використаний HTML.

# РОЗДІЛ 3. ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ ЗАСТОСУНКУ

## 3.1. Призначення застосунку

Метою вебзастосунку «BoardingTracker» є автоматизація процесу працевлаштування, що дає можливість полегшити спосіб пошуку роботи та збільшити ефективність процесу найму працівників.

Робота включає в себе:

- аналіз існуючих програмних інструментів
- ознайомлення з новими архітектурними шаблонами
- проєктування та створення програмного забезпечення для застосунку «BoardingTracker»
- тестування застосунку «BoardingTracker»

Цільова аудиторія застосунку – це компанії, які займаються пошуком та наймом нових працівників та люди, які знаходяться у пошуку роботи.

## 3.2. Цілі створення застосунку

Основними цілями створення вебзастосунку «BoardingTracker» являються:

- Пошук вакансій, перегляд наявних вакансій на ринку кандидатами.
- Створення ролі кандидата та рекрутера.
- Забезпечення взаємодії між кандидатом та компанією у форматі співбесід.
- Управління вакансіями рекрутером.

На Рисунок 10 наведена Use-Case діаграма до застосунку.

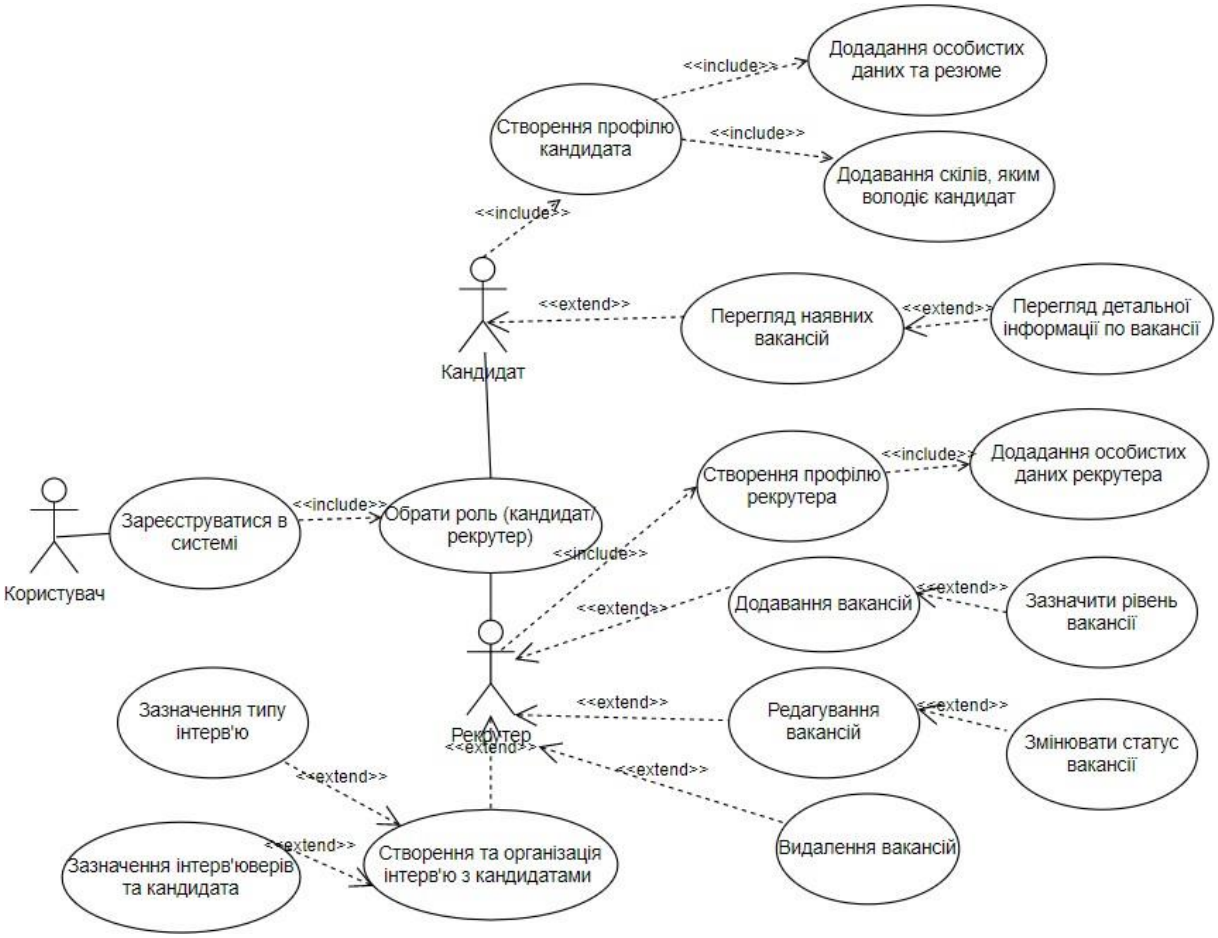


**Рисунок 10** – Use-Case діаграма

## 3.3. Вимоги до застосунку

### 3.3.1. Вимоги до застосунку в цілому

**Вимоги до структури та функціонування застосунку**

Вебзастосунок «BoardingTracker» повинен мати архітектуру, що базується на сучасних технологіях зберігання, обробки, аналізу та доступу до даних. Додаток повинен мати зрозумілий для користувача інтерфейс.

Вебзастосунок повинен бути розробленим з використанням архітектурних шаблонів, для зручного та гнучкого розширення чи змін у майбутньому.

У застосунку передбачається виділити наступні функціональні підсистеми:

- адміністративна, призначена для рекрутера, який буде додавати та редагувати вакансії у застосунку;
- підсистема користувача, призначена для кандидата, який зможе переглядати вакансії.

### 3.3.2. Вимоги до функцій, які виконуються застосунком

В розробленому вебзастосунку буде наявно дві ролі, що передбачає наявність двох підсистем з різним функціоналом. У Таблиця 1 наведено перелік функцій та задач рекрутера, а у Таблиця 2 перелік функцій та задач кандидата, що виконуються програмним способом.

**Підсистема адміністратора (рекрутера)**

**Таблиця 1** – Перелік функцій та задач рекрутера

| **Функція** | **Задача** |
|---|---|
| Управління вакансіями | Створення нової вакансії |
| | Видалення вакансії |
| | Редагування вакансії |
| | Змінювання статусу вакансії |
| Організація співбесід | Створення інтерв’ю з відповідним кандидатом та рекрутером |
| | Встановлення типу інтерв’ю |
| | Відправлення повідомлення на пошту кандидата про заплановане інтерв’ю |

**Підсистема користувача (кандидат)**

**Таблиця 2** – Перелік функцій та задач кандидата

| **Функція** | **Задача** |
|---|---|
| Пошук вакансій | Перегляд наявних вакансій |
| | Перегляд детальної інформації по вакансії |

| | |
|---|---|
| Надсилання резюме | Відправка резюме до відповідної вакансії |

Програмний продукт повинен забезпечувати коректне відображення даних у наступних браузерах актуальних версій:

- Google Chrome;
- Internet Explorer
- -Opera
- Mozilla Firefox;
- Safari.

Технічні вимоги до вебзастосунку наведено в Таблиця 3.

**Таблиця 3** – Вимоги до апаратного забезпечення вебзастосунку

| **Програмне забезпечення** | **Версія** |
|---|---|
| Операційна система | Windows |
| Мови програмування вебзастосунку | C#, TypeScript, HTML, SCSS |
| Система управління базами даних | MS SQL Server |

В застосунку передбачена валідація у разі відсутності незаповнених обов’язкових полів у застосунку і відповідне підсвідчування цих полів у користувацькому інтерфейсі.

У застосунку передбачена система авторизації та автентифікації, яка повинна базуватися на ролях (рекрутер/кандидат). Програма повинна бути покрита Unit-тестами.

# РОЗДІЛ 4. ОПИС ОРГАНІЗАЦІЇ ІНФОРМАЦІЙНОЇ БАЗИ

## 4.1. Логічна структура бази даних

В даному проєкті в якості СУБД використовується MS SQL Server. Діаграма бази даних наведена в ДОДАТОК А. Перелік всіх наявних таблиць наведено в Таблиця 4.

**Таблиця 4** – Перелік таблиць в базі даних

| Номер | Таблиця | Опис |
|---|---|---|
| 1 | **Error! Reference source not found.** | Таблиця для збереження інформації про користувачів та їх ролі |
| 2 | **Error! Reference source not found.** | Таблиця для збереження токенів, які використовуються при авторизації |
| 3 | **Error! Reference source not found.** | Таблиця для збереження даних про рекрутерів |
| 4 | **Error! Reference source not found.** | Таблиця для збереження даних про кандидатів |
| 6 | **Error! Reference source not found.** | Таблиця для збереження навиків |
| 9 | **Error! Reference source not found.** | Таблиця для збереження даних про вакансії |

| | | |
|---|---|---|
| 10 | **Error! Reference source not found.** | Таблиця-довідник в якій зберігаються статуси вакансії |
| 11 | **Error! Reference source not found.** | Таблиця-довідник в якій зберігаються типи вакасій (онлайн/офлайн/парт-тайм) |
| 12 | **Error! Reference source not found.** | Таблиця-довідник в якій зберігаються рівні вакансій(junior/middle/senior) |
| 13 | **Error! Reference source not found.** | Таблиця для збереження даних про інтерв'ю |
| 14 | **Error! Reference source not found.** | Таблиця-довідник в якій зберігаються типи інтерв'ю (HR-interview, Technical interview) |

### 4.2. Опис таблиць

Структура кожної таблиці є основою бази даних, оскільки саме в них зберігаються всі дані. Вона визначається вмістом запитів, звітів і форм, які потрібно отримати при взаємодії з базою даних. Проєктування структури таблиць є важливим кроком у розробці бази даних.

У Таблиця 5 вказані поля сутності User.

**Таблиця 5** – Поля таблиці Userss

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| Id | uniqueidentifier | Ідентифікатор |
| Email | nvarchar(50) | Електронна пошта |
| Password | nvarchar(50) | Пароль |
| Role | nvarchar(50) | Роль |

У Таблиця 6 вказані поля сутності RefreshToken.

**Таблиця 6** – Поля таблиці RefreshTokens

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| RefreshTokenGID | uniqueidentifier | Ідентифікатор |
| Token | nvarchar(MAX) | Токен |
| UserId | uniqueidentifier | Ідентифікатор користувача |

У Таблиця 7 вказані поля сутності Recruiter.

**Таблиця 7** – Поля таблиці Recruiters

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| Id | uniqueidentifier | Ідентифікатор |
| FirstName | nvarchar(50) | Ім’я |
| LastName | nvarchar(50) | Прізвище |
| ProfileImageUrl | nvarchar(MAX) | URL фото |
| UserId | uniqueidentifier | Ідентифікатор користувача |

У Таблиця 8 вказані поля сутності Candidate.

**Таблиця 8** – Поля таблиці Candidates

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| Id | uniqueidentifier | Ідентифікатор |
| FirstName | nvarchar(50) | Ім’я |
| LastName | nvarchar(50) | Прізвище |
| PhoneNumber | nvarchar(50) | Телефон |
| Biography | nvarchar(MAX) | Короткий опис про себе |
| ResumeUrl | nvarchar(MAX) | Url резюме |
| UserId | uniqueidentifier | Ідентифікатор користувача |

У Таблиця 9 вказані поля сутності Skill.

**Таблиця 9** – Поля таблиці Skills

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| Id | int | Ідентифікатор |
| Name | nvarchar(50) | Назва навички |

У Таблиця 10 вказані поля сутності Vacancy.

**Таблиця 10** – Поля таблиці Vacancies

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| Id | Int | Ідентифікатор |
| Title | nvarchar(50) | Назвва вакансії |
| Description | nvarchar(MAX) | Опис вакансії |
| Salary | decimal(18, 0) | Зарплата |
| SeniorityLevelId | Int | Рівень вакансії |
| VacancyTypeId | Int | Тип вакансії |
| VacancyStatusId | Int | Статус вакансії |
| RecruiterId | uniqueidentifier | Рекрутер, який створив вакансію |

У Таблиця 11 вказані поля сутності VacancyStatus.

**Таблиця 11** – Поля таблиці VacancyStatuses

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| Id | int | Ідентифікатор |
| Name | Nvarchar(50) | Назва статусу |

У Таблиця 12 вказані поля сутності VacancyType.

**Таблиця 12** – Поля таблиці VacancyTypes

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| Id | int | Ідентифікатор |
| Name | Nvarchar(50) | Назва типу |

У Таблиця 13 вказані поля сутності SeniorityLevel.

**Таблиця 13** – Поля таблиці SeniorityLevels

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| Id | int | Ідентифікатор |
| Name | Nvarchar(50) | Назва рівня |

У Таблиця 14 вказані поля сутності

Interview.

**Таблиця 14** – Поля таблиці Interviews

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| Id | Int | Ідентифікатор |
| Title | Nvarchar(50) | Назва |
| StartTime | datetime | Початок інтерв’ю |
| EndTime | datetime | Кінець інтерв’ю |
| VacancyId | int | Вакансія |
| RecruiterId | uniqueidentifier | Рекрутер |
| CandidateId | uniqueidentifier | Кандидат |
| InterviewTypeId | int | Тип інтерв’ю |

У Таблиця 15 вказані полясутності InterviewType.

**Таблиця 15** – Поля таблиці InterviewTypes

| **Атрибут** | **Тип** | **Опис** |
|---|---|---|
| Id | int | Ідентифікатор |
| Name | Nvarchar(50) | Назва типу інтерв’ю |

# РОЗДІЛ 5. РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

Код застосунку зберігається у GitHub репозиторії (https://github.com/Soniabel/BoardingTracker.git). Розробка виконана у три етапи, спочатку була створена база даних, потім було розроблено серверну частину застосунку, і в результаті на основі готових кінцевих точок було реалізовано клієнтську частину.

## 5.1. Реалізація серверної частини

В ході проєктування програми було використано різні архітектурні шаблони. Для забезпечення високорівневої абстракції та розділення обов'язків програми було використано Onion архітектуру (див. Рисунок 11 – *Реалізація* Onion архітектури

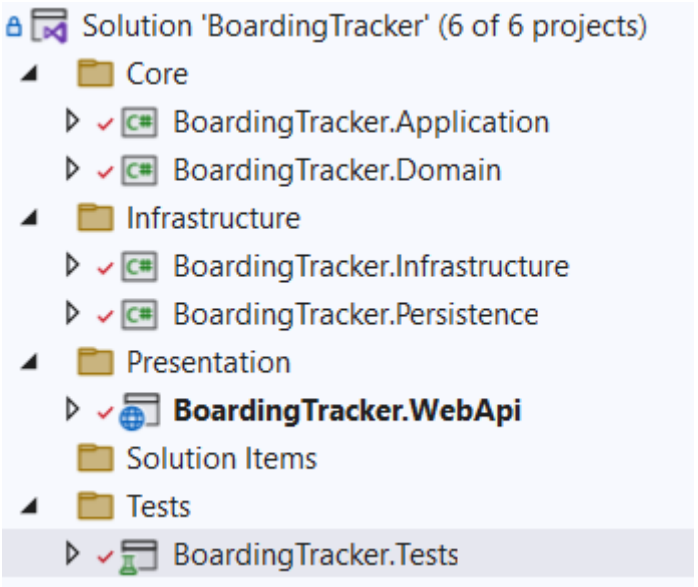


**Рисунок 11** – Реалізація Onion архітектури

Серверна частина застосунку була виконана з використанням технології ASP.NET Web API і всі запити в програмі відповідають архітектурному стилю RESTful API.

Щоб забезпечити гнучкість та ефективність при проєктуванні програмного забезпечення було використано шаблон CQRS (див. Рисунок 12). У результаті це дозволило зменшити складність коду, так як через розбиття на папки стало доволі зручно орієнтуватися в коді та швидко вносити зміни у потрібні частини коду.

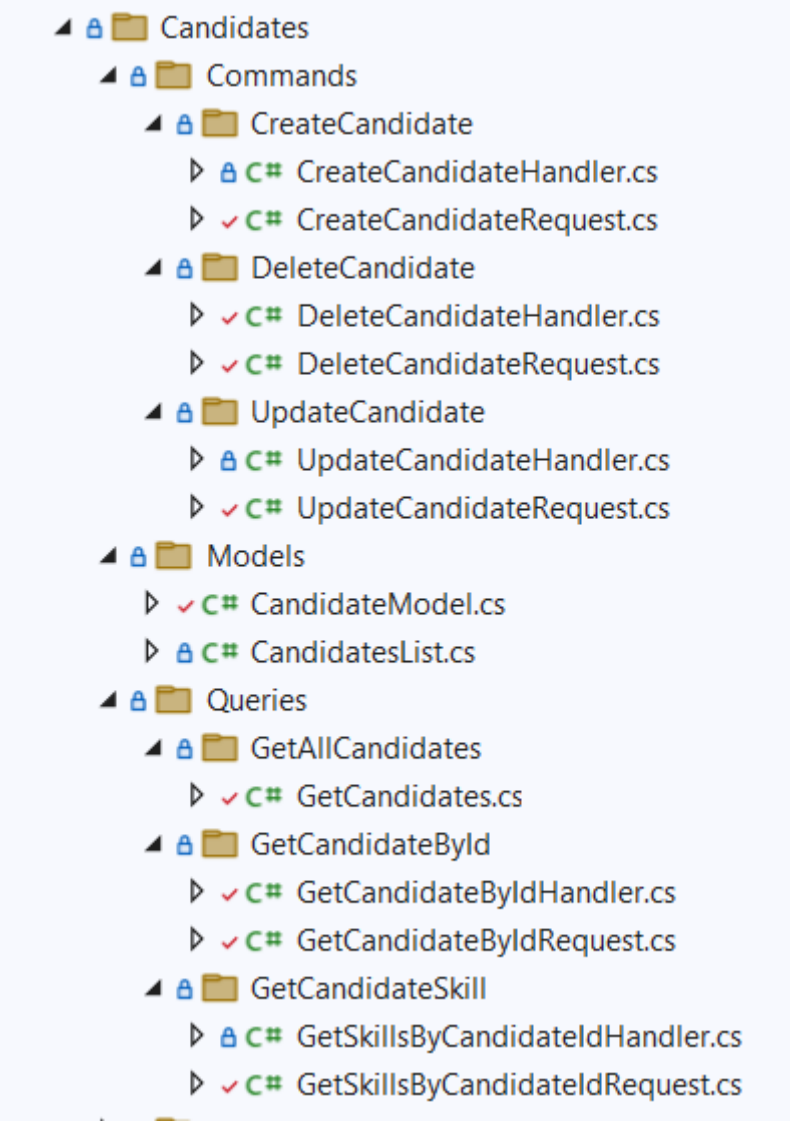


**Рисунок 12** – Структура CQRS

Для кожної з моделей у застосунку реалізовано валідацію за допомогою бібліотеки для валідації даних FluentValidation. Дана бібліотека дозволяє забезпечити валідацію на рівні бізнес-логіки. На Рисунок 13 наведено один з методів, в якому була зазначена валідація.

```
using BoardingTracker.Application.Candidates.Commands.CreateCandidate;
using FluentValidation;

namespace BoardingTracker.Application.Candidates.Validators
{
    3 references
    public class CreateCandidateValidator : AbstractValidator<CreateCandidateRequest>
    {
        2 references
        public CreateCandidateValidator()
        {
            RuleFor(candidate => candidate.FirstName).NotEmpty().NotNull().MaximumLength(50);
            RuleFor(candidate => candidate.LastName).NotEmpty().NotNull().MaximumLength(50);
            RuleFor(candidate => candidate.PhoneNumber).NotEmpty().NotNull().MaximumLength(50);
            RuleFor(candidate => candidate.Biography).NotEmpty().NotNull();
            RuleFor(candidate => candidate.ResumeFileName).NotEmpty().NotNull();
        }
    }
}
```

**Рисунок 13** – Реалізація валідації

Дії користувача обробляються контролерами. Вони знаходяться у проєкті Web API. Так як робота виконана за шаблоном CQRS, щоб організувати взаємодію між класами та компонентами застосунка був використаний

шаблон Посередник, а конкретніше бібліотека MediatR. Приклад реалізації контролера наведено на Рисунок 14.

```
[HttpPut]
[ProducesResponseType(StatusCodes.Status200OK, Type = typeof(CandidatesList))]
0 references
public async Task<IActionResult> UpdateCandidate([FromBody] UpdateCandidateRequest updateCandidateRequest)
{
    var result = await Mediator.Send(updateCandidateRequest);
    return Ok(result);
}

[HttpGet]
[ProducesResponseType(StatusCodes.Status200OK, Type = typeof(CandidatesList))]
0 references
public async Task<IActionResult> Get()
{
    var result = await Mediator.Send(new GetCandidates());
    return Ok(result);
}

[HttpGet("candidatebyuserid")]
[ProducesResponseType(StatusCodes.Status200OK, Type = typeof(CandidatesList))]
0 references
public async Task<IActionResult> GetById(string userId)
{
    var result = await Mediator.Send(new GetCandidateByIdRequest { UserId = Guid.Parse(userId) });
    return Ok(result);
}

[HttpGet("skills/{id}")]
[ProducesResponseType(StatusCodes.Status200OK, Type = typeof(SkillList))]
0 references
public async Task<IActionResult> SkillListByCandidateId(string userId)
{
    var result = await Mediator.Send(new GetSkillsByCandidateIdRequest { UserId = Guid.Parse(userId) });
    return Ok(result);
}
```

**Рисунок 14** – Реалізація контролера

Для обробки винятків в проєкті використано глобальну обробку помилок за допомого вбудованого ПЗ проміжного рівня (Global Exception Middleware).

Даний функціонал дозволяє справлятися з непередбачуваними винятками в застосунку (див. Рисунок 15).

```
public class GlobalExceptionHandlingMiddleware
{
    private readonly RequestDelegate _next;
    private readonly ILogger<GlobalExceptionHandlingMiddleware> _logger;
    private readonly IWebHostEnvironment _environment;

    0 references
    public GlobalExceptionHandlingMiddleware(
        RequestDelegate next, ILogger<GlobalExceptionHandlingMiddleware> logger, IWebHostEnvironment environment)
    {
        _next = next;
        _logger = logger;
        _environment = environment;
    }

    0 references
    public async Task InvokeAsync(HttpContext context)
    {
        try
        {
            await _next(context);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, ex.Message);
            var responseObj = HandleException(ex, out var statusCode);
            context.Response.ContentType = "application/json";
            context.Response.StatusCode = statusCode;

            await context.Response.Body.WriteAsync(
                JsonSerializer.SerializeToUtf8Bytes(
                    responseObj, new JsonSerializerOptions { IgnoreNullValues = true }));
        }
    }

    1 reference
    private ExceptionModel HandleException(Exception ex, out int statusCode)
    {
        var errorResponse = new ExceptionModel
        {
            Message = ex.Message
        };
```

**Рисунок 15** – Реалізація Global Exception Handler

В застосунку реалізований функціонал присилання повідомлення на пошту кандидата, якщо рекрутер організував інтерв’ю, для цього функціоналу було використано хмарну платформу SendGrid, яка надає функціонал для надсилання листів (див. Рисунок 16).

```
0 references
public static class ServiceCollectionExtension
{
    1 reference
    public static void AddCustomSendGrid(this IServiceCollection services, IConfiguration configuration)
    {
        var sendGridModel = new SendGridModel();
        services.AddSendGrid(options =>
        {
            options.ApiKey = configuration.GetSection("SendGridEmailSettings")
            .GetValue<string>("APIKey");
            configuration.Bind("SendGridEmailSettings", sendGridModel);
        });

        services.AddSingleton(sendGridModel);

        services.AddScoped<IEmailSender, EmailSender>();
    }
}
```

**Рисунок 16** – Підключення Sengrid до проєкту

В результаті реалізовані всі запити CRUD та підключене відкрите програмне забезпечення Swagger. Він надає набір інструментів для створення специфікацій API у форматі OpenAPI. Цей формат описує доступні кінцеві точки, параметри, формати даних, схеми запитів та відповідей API (див. Рисунок 17).

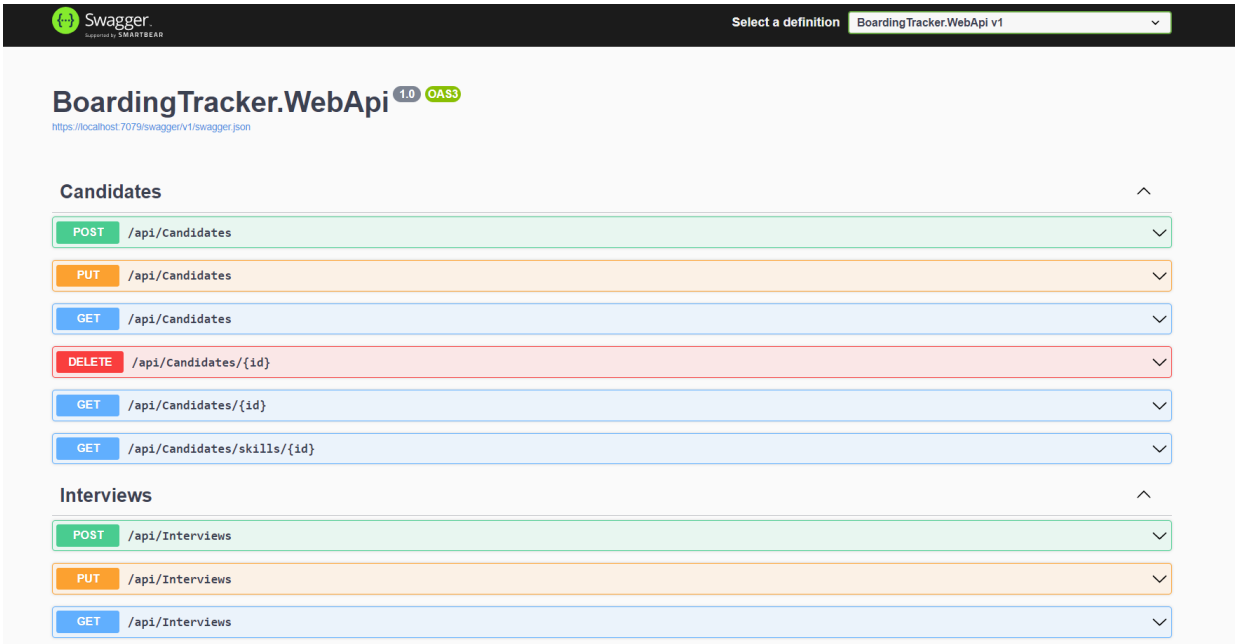


**Рисунок 17** – Кінцеві точки в Swagger

## 5.2. Реалізація клієнтської частини

Клієнтська частина відокремлена від основної логіки на сервері. Для реалізації фронтенду було використано Angular, який містить в собі модульну архітектуру. Реалізація модульної архітектури у вебзастосунку наведено на Рисунок 18.

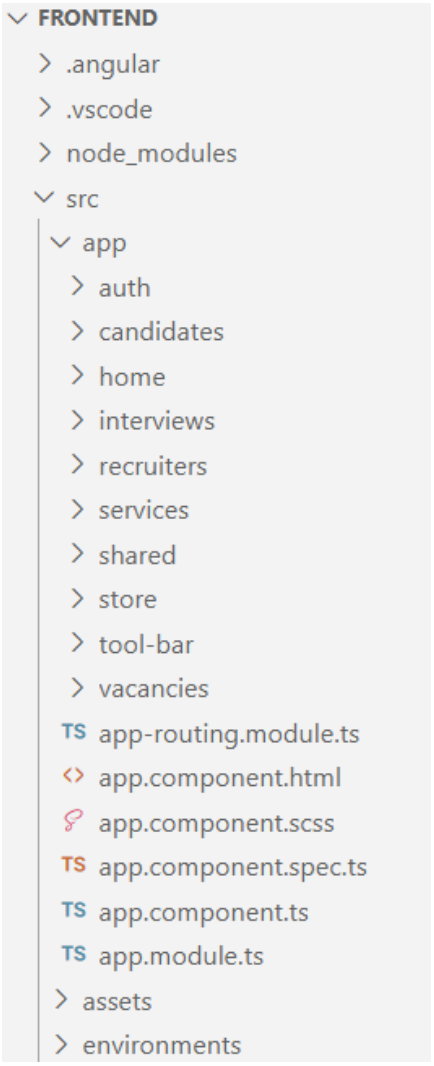


**Рисунок 18** – Модульна архітектура застосунку

Застосунок розбитий на окремі папки відповідно до логіки, так у папці candidates зберігається модель, компонента, модуль до даної сутності. Структура файлів у папці наведено на Рисунок 19.

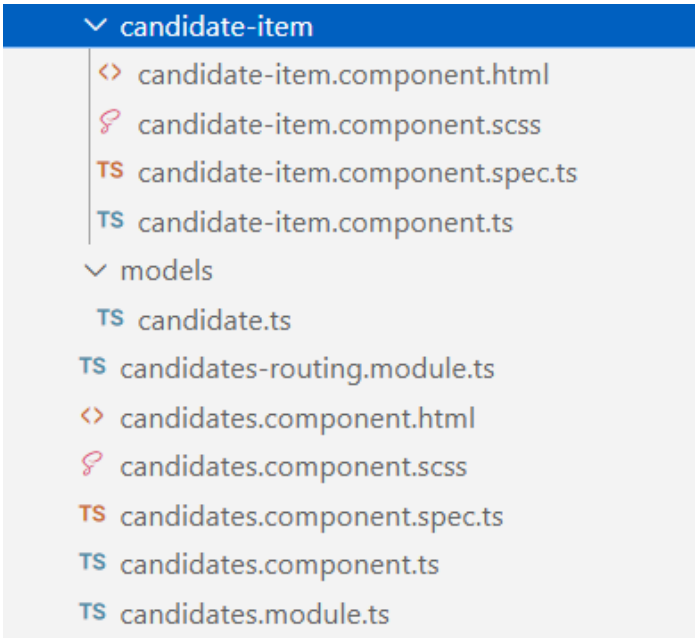


**Рисунок 19** – Структура файлів

Всі http запити на сервер зберігаються в папці services. В кожному сервісі зазначається з якою саме кінцевою точкою потрібно зв'язати відповідну компоненту. Нижче наведено один із запитів на сервер для отримання всіх кандидатів (див. Рисунок 20)

```
export class CandidateService {

  constructor(private httpClient: HttpClient) { }

  public getAll(): Observable<Candidate[]> {
    return this.httpClient.get<ResponseModel<Candidate[]>>(`${environment.baseUrl}/candidates`)
    .pipe(map((data) => {
      return data.items
    }));
  }

  public getCandidateSkills(userId: string): Observable<Skill[]> {
    return  this.httpClient.get<ResponseModel<Skill[]>>(`${environment.baseUrl}/candidates/skills/${userId}`)
      .pipe(map((data) => {
        return data.items
      }));
  }
}
```

**Рисунок 20** – Реалізація сервісу

### 5.3. Тестування вебзастосунку

Велика частина коду покрита unit-тестами. Для реалізації автоматизованих тестів було використано бібліотеку Xunit [14]. Протестовано

кожну CRUD операцію та валідацію у застосунку. Приклад реалізованого тесту наведено на Рисунок 21.

```
using Xunit;

namespace BoardingTracker.Tests.VacancyStatuses.Commands
{
    0 references
    public abstract class CreateVacancyStatusTests
    {
        2 references
        public abstract class CreateVacancyStatusTest : BaseTest
        {
            protected readonly CreateVacancyStatusRequest _vacancyStatusRequest;

            protected readonly CreateVacancyStatusHandler _vacancyStatusHandler;

            0 references
            protected CreateVacancyStatusTest()
            {
                _vacancyStatusRequest = new CreateVacancyStatusRequest()
                {
                    Name = "TestName"
                };

                _vacancyStatusHandler = new CreateVacancyStatusHandler(_dbContext, _mapper);
            }
        }

        0 references
        public class Handle : CreateVacancyStatusTest
        {
            [Fact]
            0 references
            public async Task VacancyStatus_model_is_returned_when_request_is_valid()
            {
                var expectedVacancyStatus = new VacancyStatusModel
                {
                    Id = 3,
                    Name = "TestName"
                };
                var result = await _vacancyStatusHandler.Handle(_vacancyStatusRequest, new CancellationToken());

                result.Should().BeEquivalentTo(expectedVacancyStatus);
            }
```

**Рисунок 21** – Реалізація тесту

Кількість тестів – 166, результати тестів, які пройшли успішно наведено на Рисунок 22**Рисунок *22***.

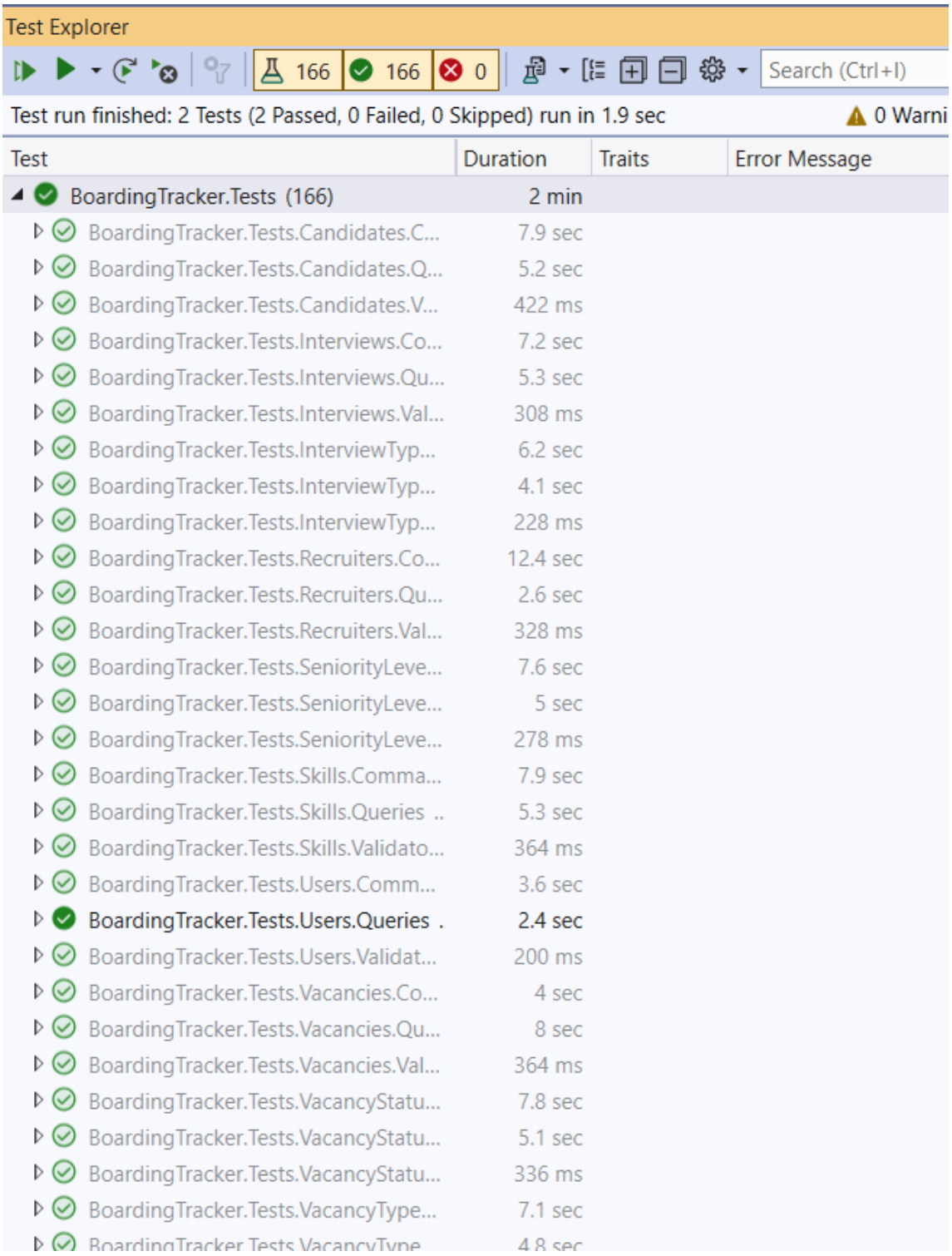


**Рисунок 22** – Результати тестів на серверній частині

На клієнтській частині застосунку теж було проведено модульне тестування за допомогою фреймворка Jasmine. Для автоматизованого запуску та виконання тестів було використано тестовий ранер Karma. На Рисунок 23

наведено приклад тестування компоненти Candidate. На Рисунок 23 наведено результати проведених тестів.

```
import { HttpClientTestingModule } from '@angular/common/http/testing';
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { ReactiveFormsModule } from '@angular/forms';
import { MatDialogModule } from '@angular/material/dialog';

import { CandidatesComponent } from './candidates.component';

describe('CandidatesComponent', () => {
  let component: CandidatesComponent;
  let fixture: ComponentFixture<CandidatesComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [ ReactiveFormsModule, HttpClientTestingModule, MatDialogModule ],
      declarations: [ CandidatesComponent ]
    })
    .compileComponents();

    fixture = TestBed.createComponent(CandidatesComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});

```

**Рисунок 23** – Реалізація тестування компоненти

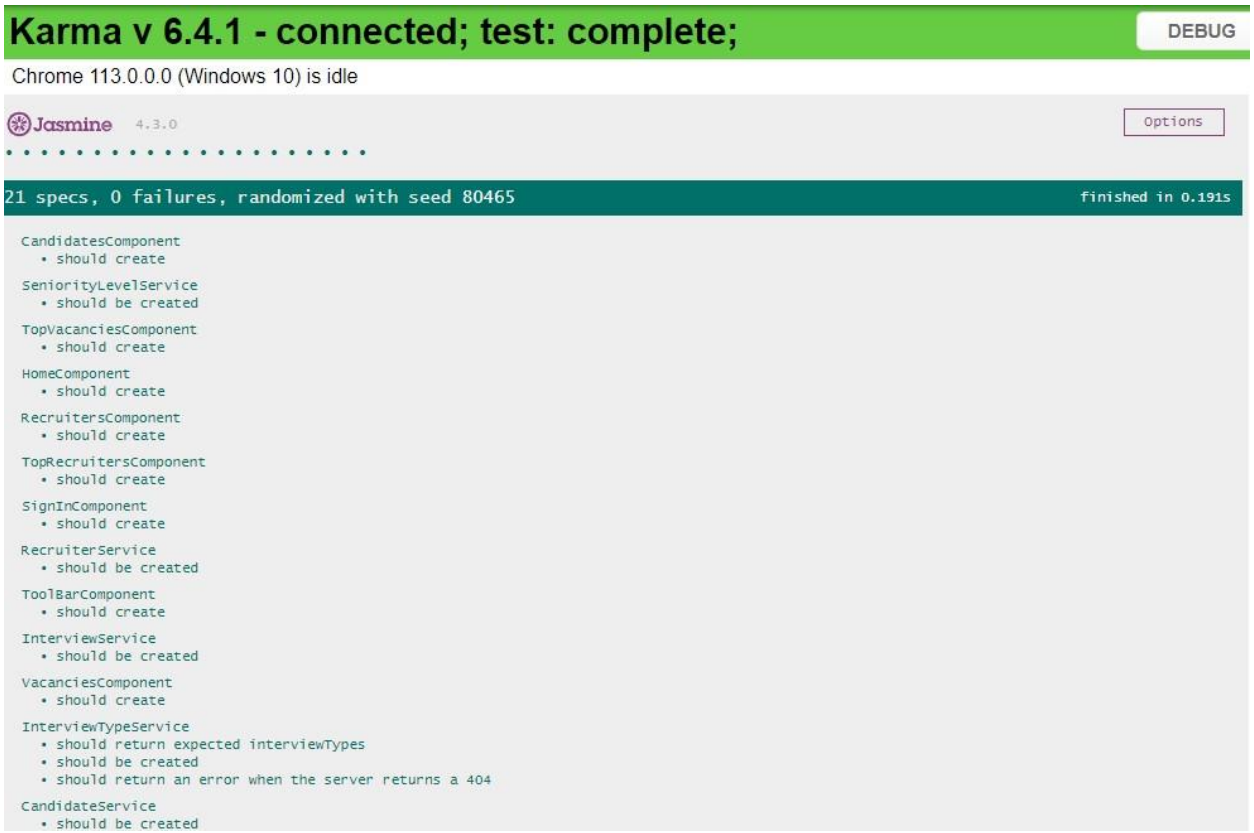


**Рисунок 24** – Результати тестів на клієнтській частині

# РОЗДІЛ 6. ІНСТРУКЦІЯ КОРИСТУВАЧА

Під час запуску вебзастосунку користувачу відкривається головна сторінка застосунку, на якій показано топ рекрутерів та вакансій. Головна сторінка наведена на Рисунок 25.

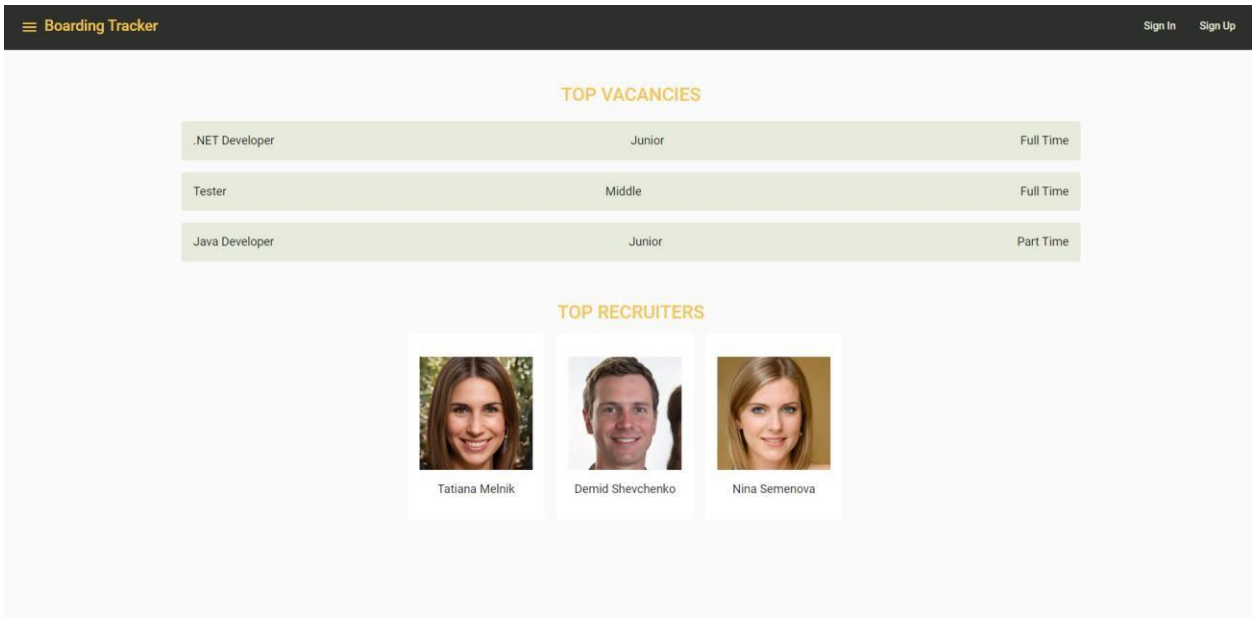


**Рисунок 25** – Головна сторінка

Далі користувачу пропонується авторизуватися або зареєструватися. При реєстрації користувач повинен обрати свою роль кандидат або рекрутер і в залежності від вибору, користувачу буде доступний різний функціонал застосунка (див. Рисунок 26).

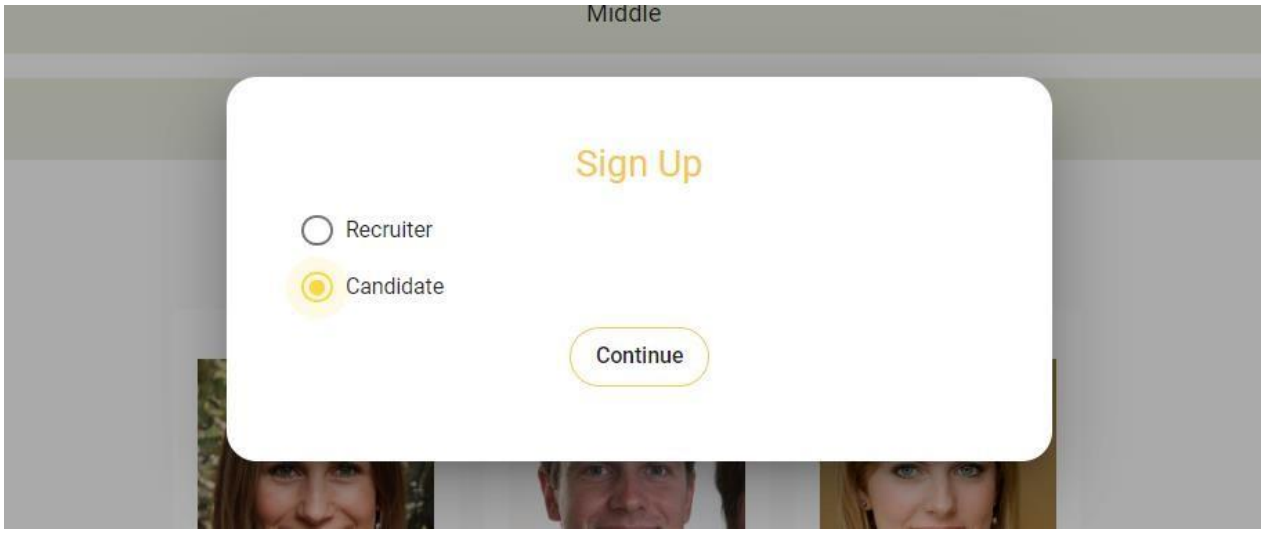


**Рисунок 26** – Реєстрація, вибір ролі

При виборі ролі кандидата, користувачу потрібно заповнити такі поля, які наведені на Рисунок 27. При виборі ролі – рекрутер, поля будуть відрізнятися, реєстраційна форма для рекрутера наведена на Рисунок 28.

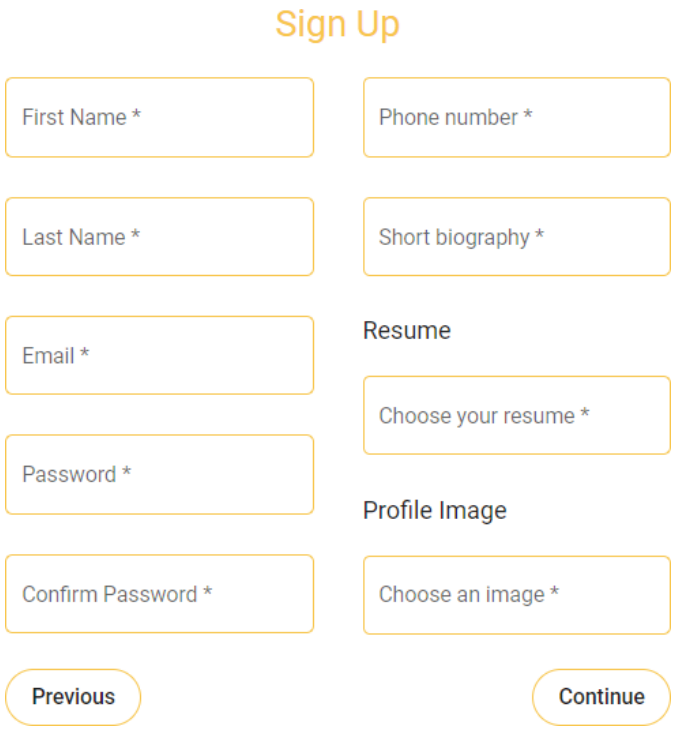


**Рисунок 27** – Поля реєстрації для кандидата

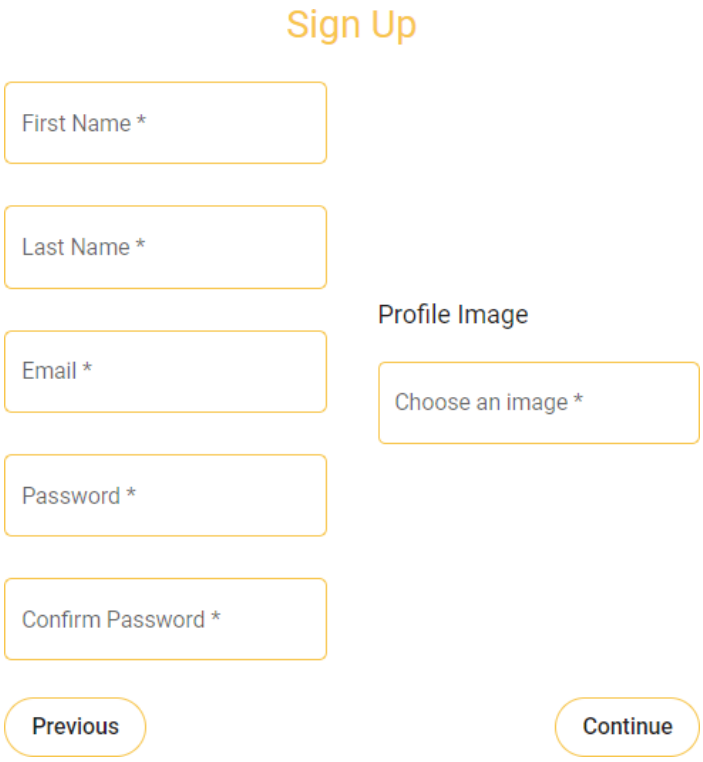


**Рисунок 28** – Поля реєстрації для рекрутера

При автентифікації користувача як кандидата, йому доступне меню, де наявна сторінка Вакансій (див. Рисунок 29**Рисунок *29***).

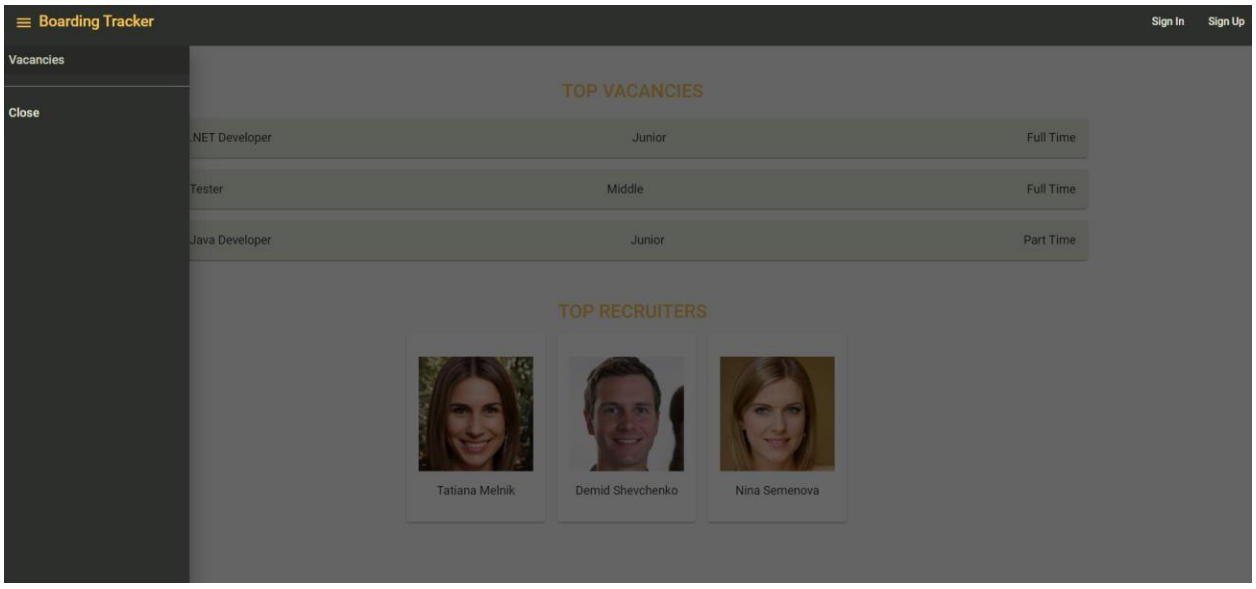


**Рисунок 29** – Меню кандидата

При натисканні на кнопку вакансій, кандидат переходить на сторінку доступних вакансій (див. Рисунок 30) та при натисканні на відповідну вакансію може переглянути детальну інформацію про вакансію та відгукнутись на неї. (див. Рисунок 31).


**Рисунок 30** – Сторінка з доступними вакансіями

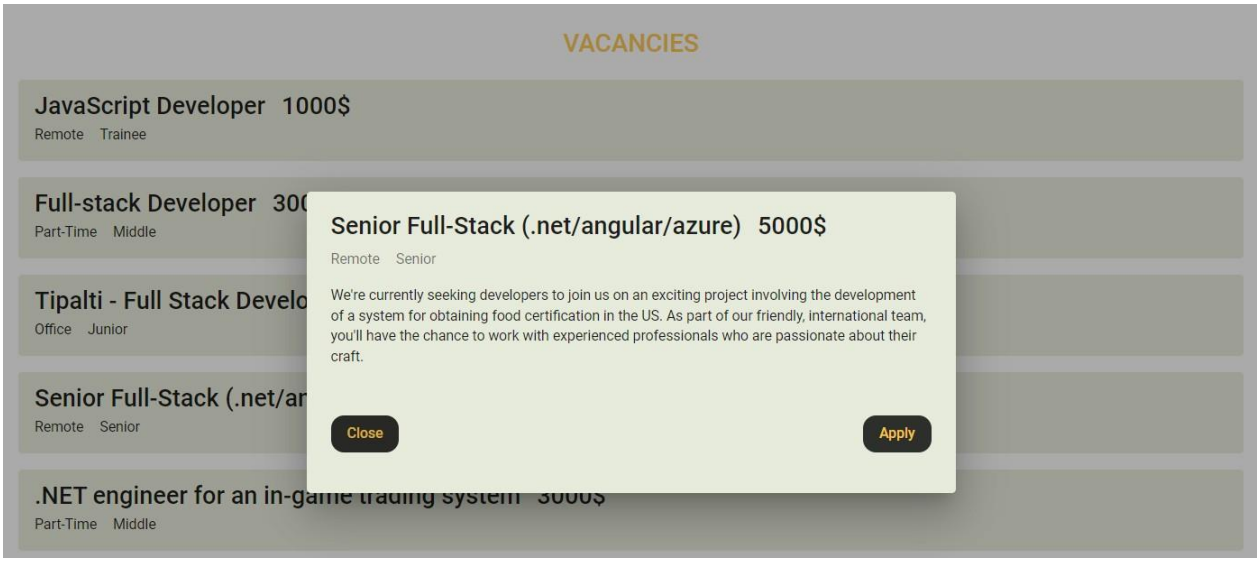


**Рисунок 31** – Детальна інформація про вакансію

При автентифікації користувача як рекрутера, йому доступне меню, де наявні сторінки Вакансій, Інтерв'ю, Кандидатів (див. Рисунок 32).

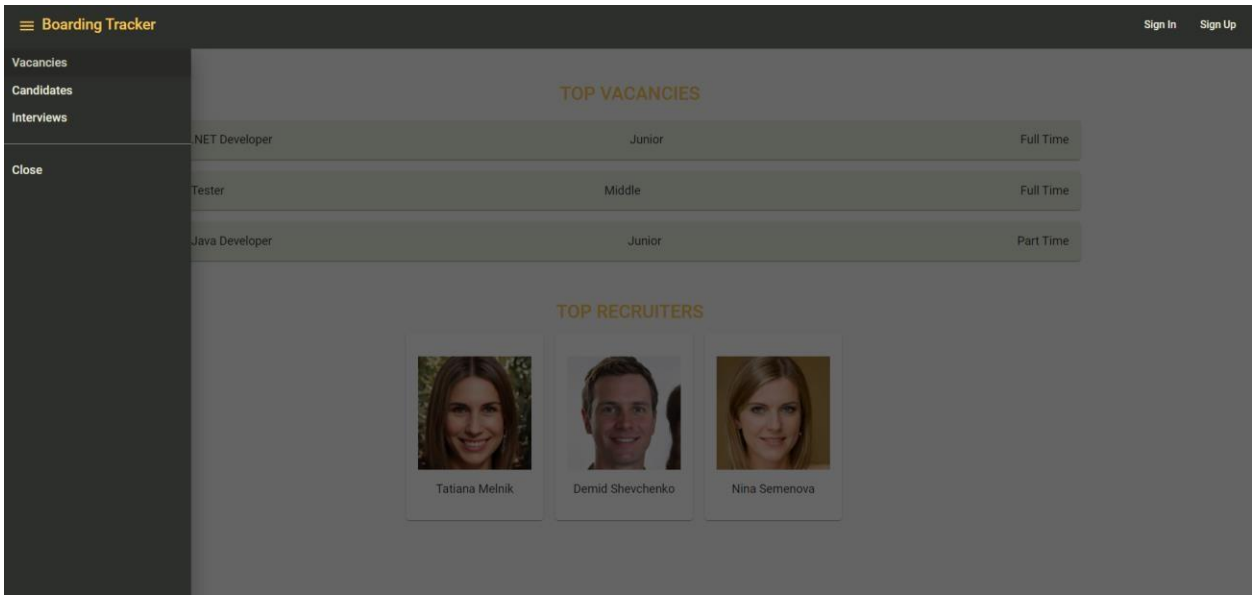


**Рисунок 32** – Меню рекрутера

При переході на сторінку вакансій, рекрутер може переглядати всі створені вакансії та додати нові. Натиснувши кнопку додавання, користувач переходить на сторінку додавання (див. Рисунок 33) .Для кожного поля наявна валідація, якщо користувач не заповнить якесь з полів вони підсвітяться червоним і вакансія не створиться. (див. Рисунок 34**Рисунок *34***).

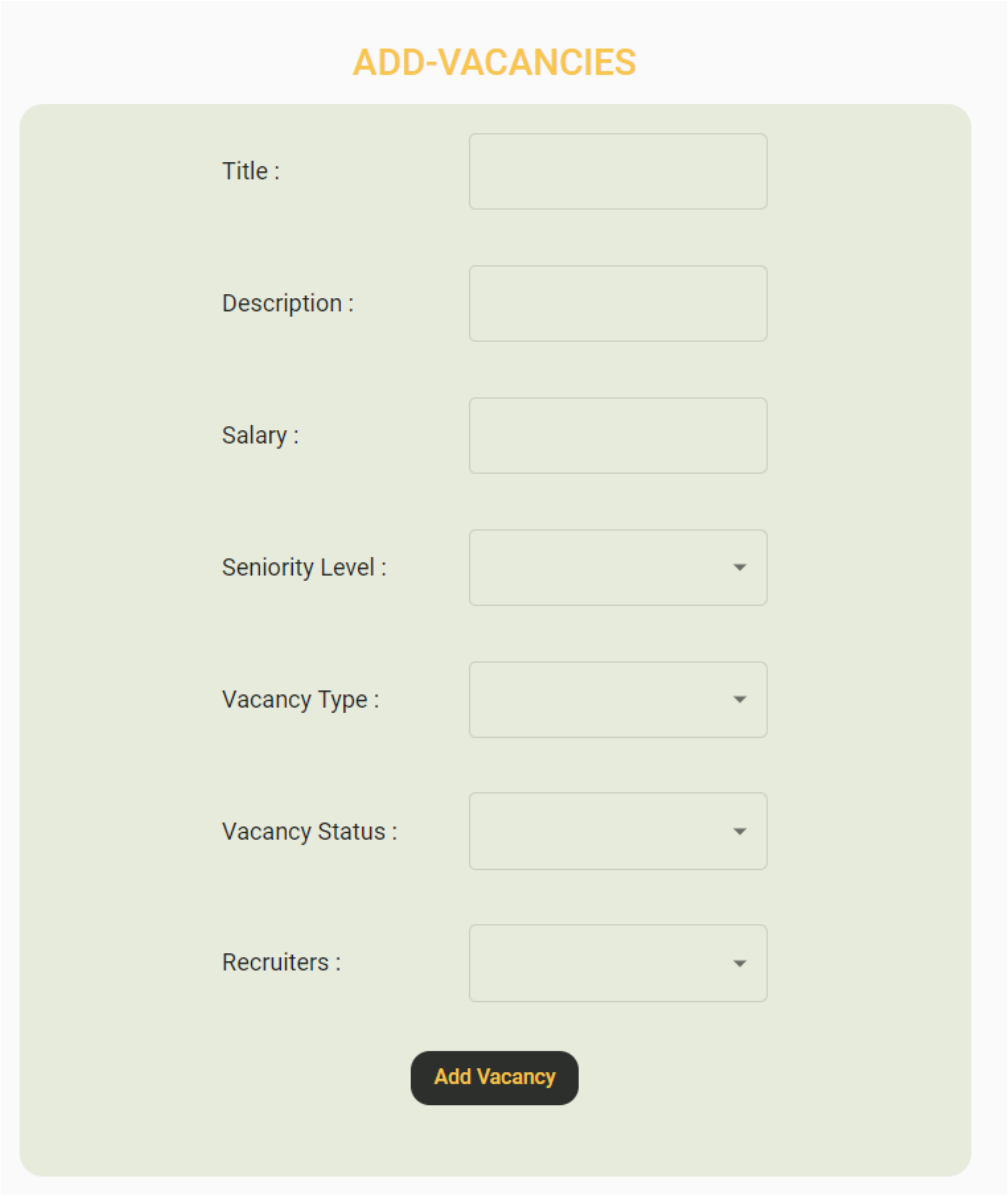


**Рисунок 33** – Сторінка додавання вакансій

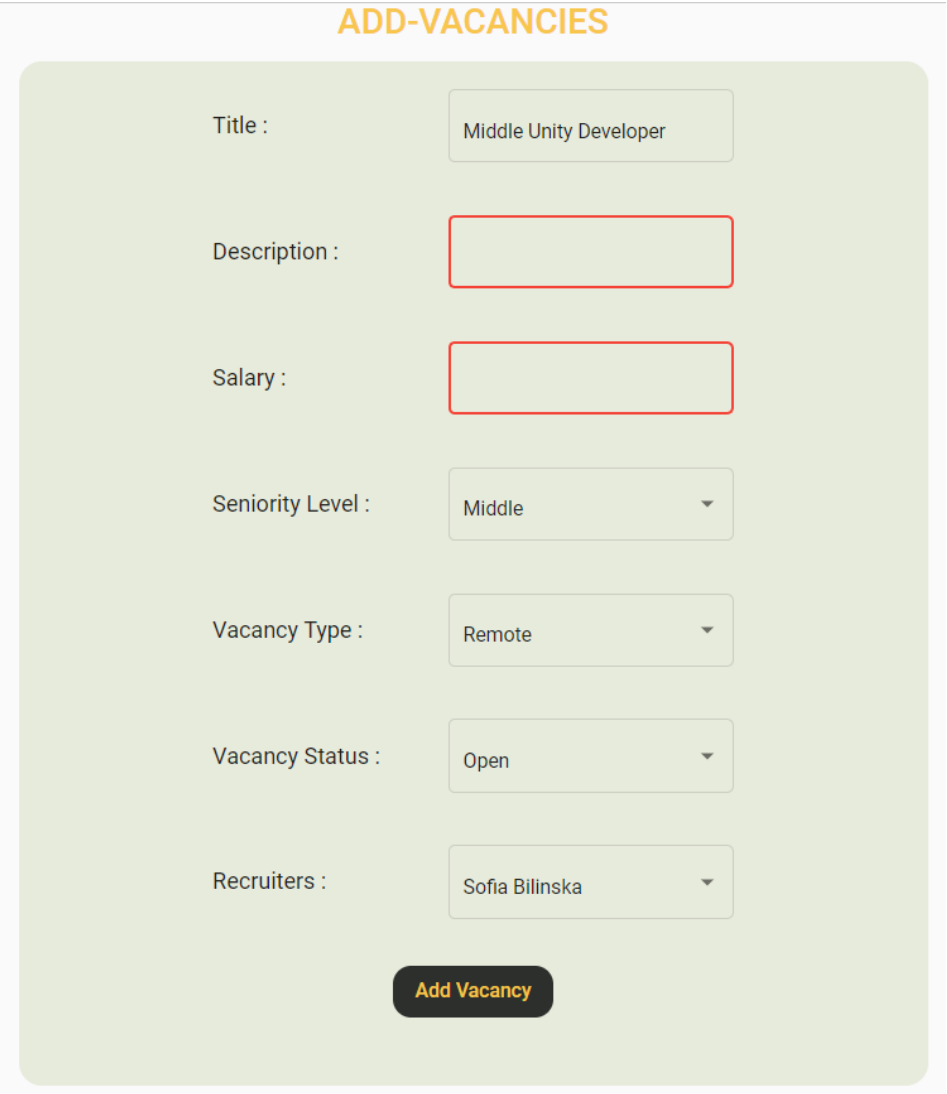


**Рисунок 34** – Перевірка на незаповнені поля

При натисканні на сторінку кандидатів, користувачу доступний список всіх кандидатів (див. Рисунок 35) та при натисканні на одного з них, переглянути детальну інформацію та резюме кандидата (див. Рисунок 36).

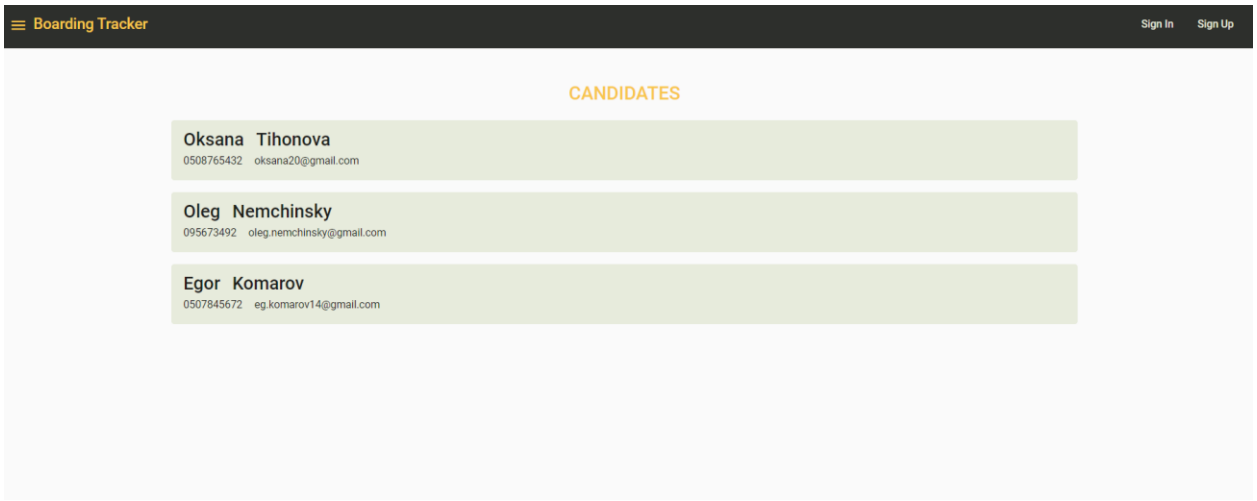


**Рисунок 35** – Сторінка Кандидатів

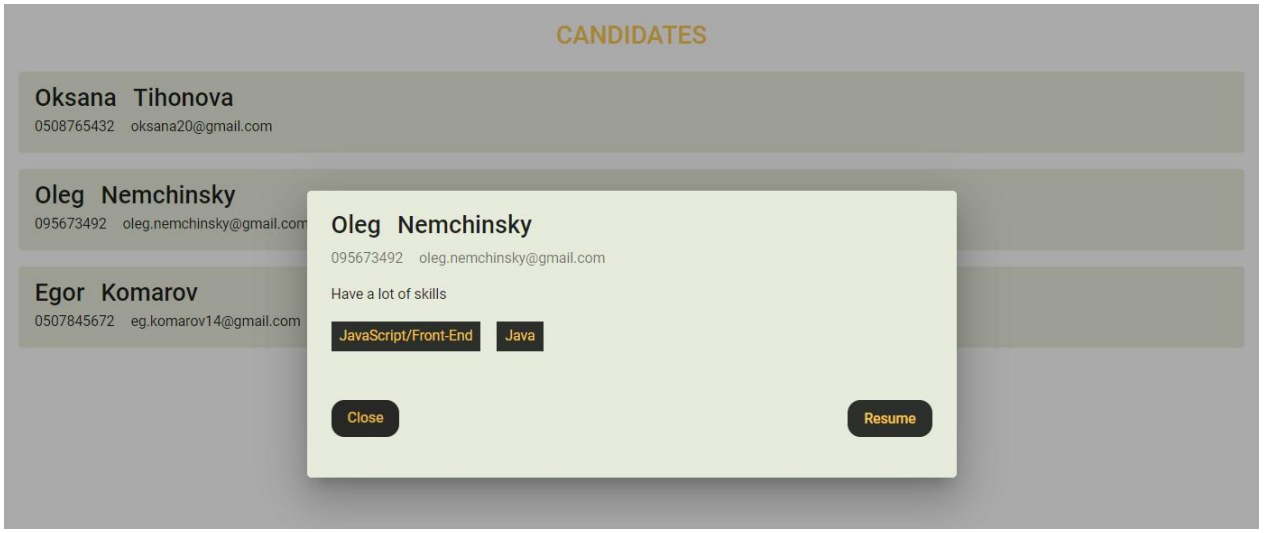


**Рисунок 36** – Детальна інформація про кандидата

Коли користувач перейде на сторінку Інтерв'ю, то на ній буде доступний весь список запланованих інтерв'ю (див. Рисунок 37), та можливість створити нове інтерв'ю з відповідним кандидатом та по відповідній вакансії (див. Рисунок 38**Рисунок *38***). Також наявна можливість створити інтерв'ю при перегляді однієї з вакансій (див. Рисунок 39).

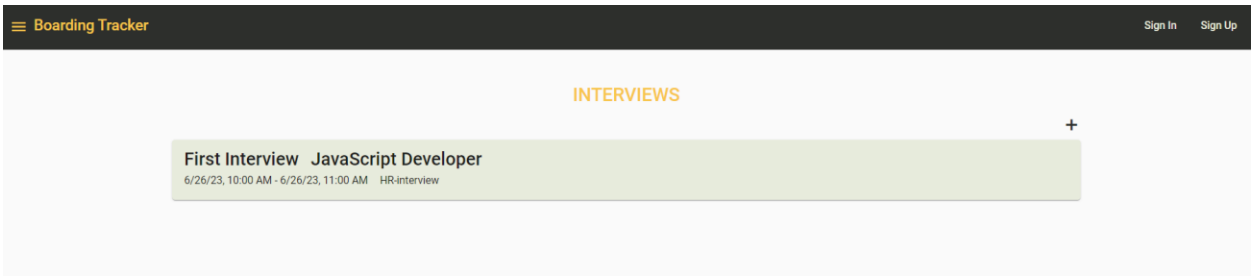


**Рисунок 37** – Сторінка Інтерв'ю

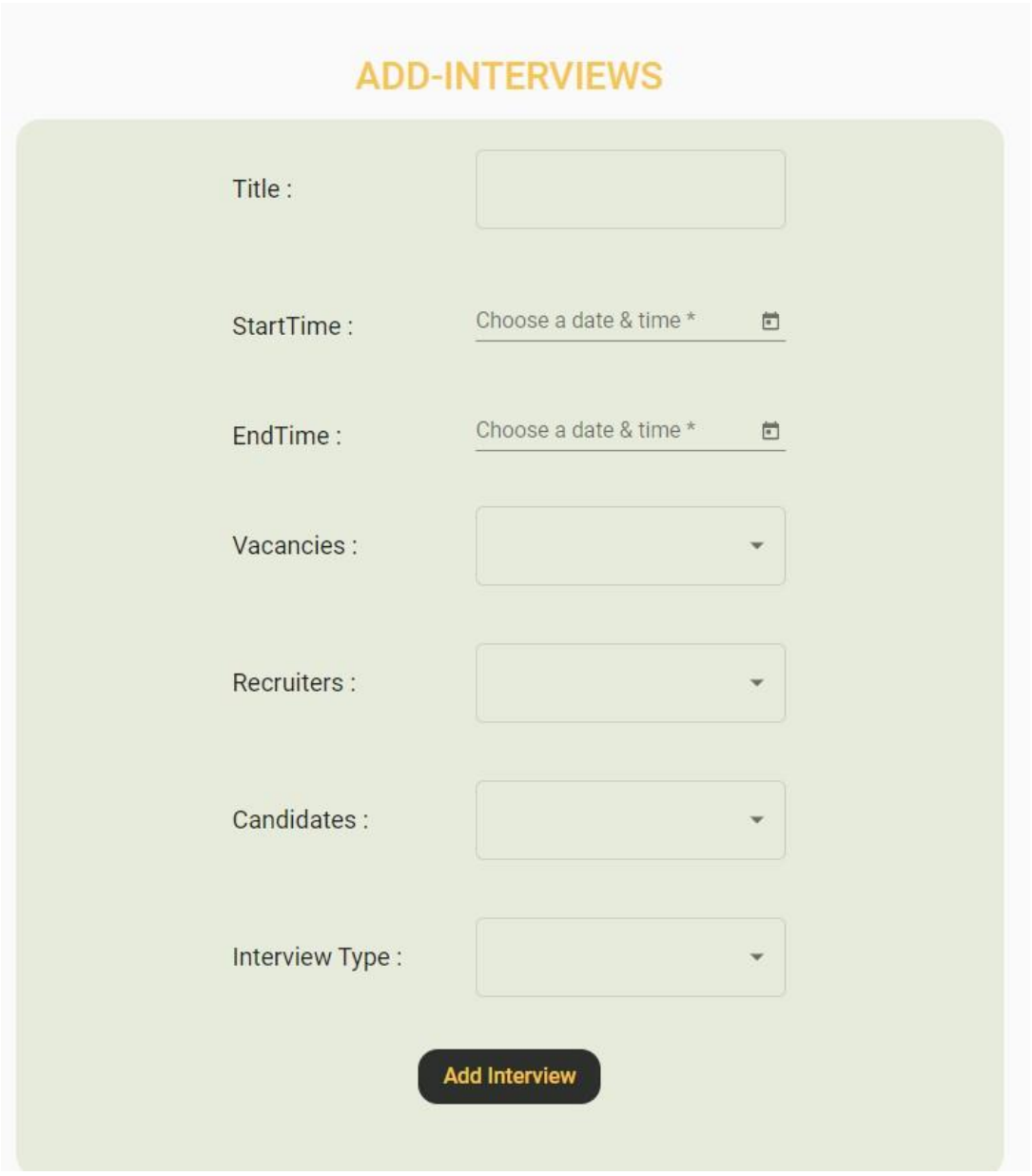


**Рисунок 38** – Сторінка додавання інтерв'ю

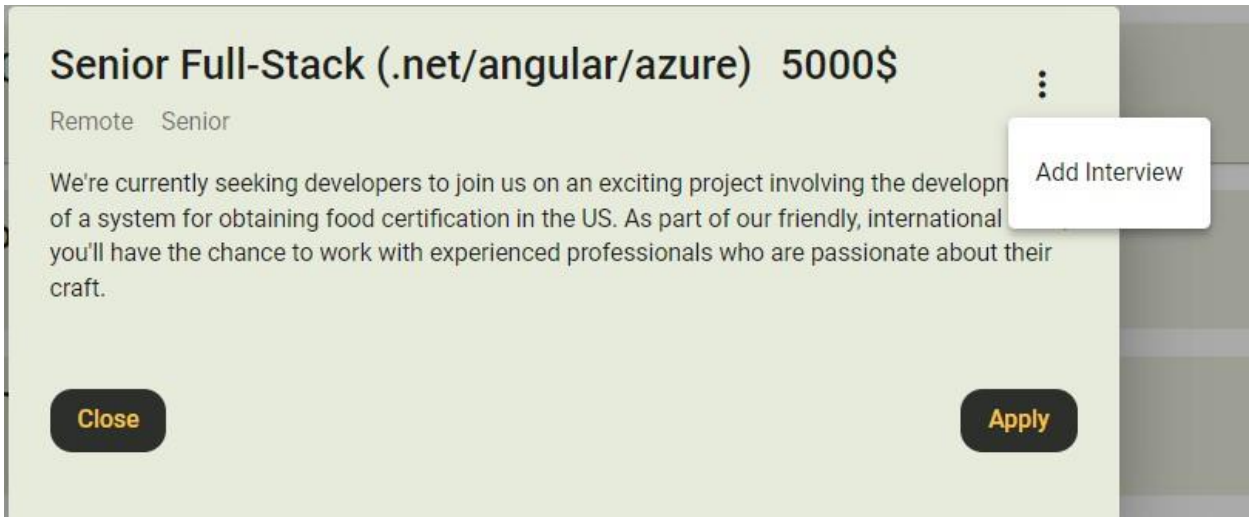


**Рисунок 39** – Додавання інтерв'ю до вакансії

При створенні інтерв’ю кандидату приходить повідомлення на пошту, яке сповіщає про заплановане інтерв’ю (див. Рисунок 40).

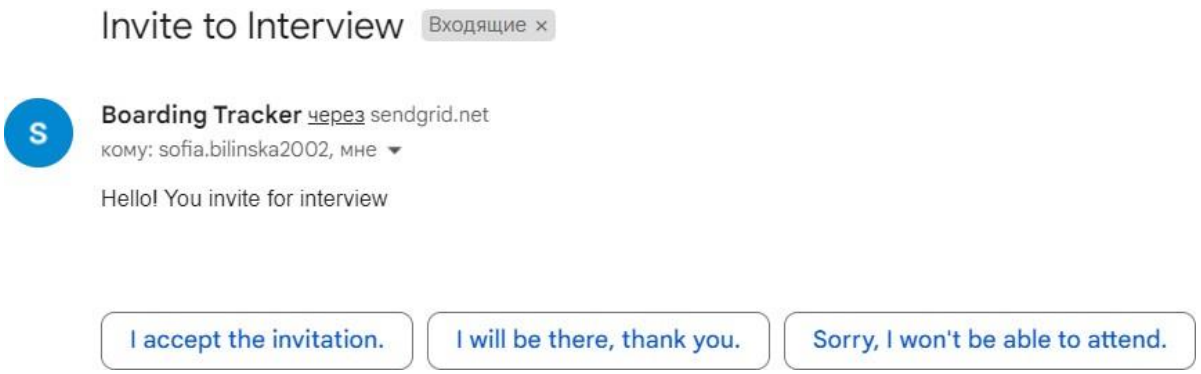


**Рисунок 40** – Запрошення на інтерв'ю

# ВИСНОВКИ

Метою даної кваліфікаційної роботи було проєктування та розробка програмного продукту для автоматизації процесу працевлаштування, що дає можливість полегшити спосіб пошуку роботи та збільшити ефективність процесу найму працівників.

Для досягнення цієї мети було розглянуто наявні існуючі системи для автоматизацій HR-процесів, досліджено різні технології для проєктування та розробки застосунків. Апробовано різні архітектурні шаблони, які зробили вебзастосунок більш гнучким.

Спроєктовано та реалізовано базу даних, на основі якої відбувалась розробка серверної частини, а для написання серверної частини було обрано ASP.NET Core Web API. Розроблено дизайн застосунку та створено клієнтську частину за допомогою фреймворка Angular. Також проведено тестування клієнтської та серверної частини застосунку.

Розроблений програмний продукт може застосовуватись компаніями, які займаються пошуком та наймом нових працівників та людьми, які знаходяться у пошуку роботи.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ


1. Цифрові компетенції як умова формування якості людського капіталу (2019): аналіт. зап. / [В. С. Куйбіда, О. М. Петроє, Л. І. Федулова, Г. О. Андрощук]. Київ: НАДУ. 28 с.
2. Нагибіна Н. І., Щукина А. А. (2017). HR-Digital: цифрові технології в управлінні людськими ресурсами. Науковедення: інтернет-журнал, Т. 9. 17.
3. Рудакова С. Г., Данилевич Н. С., Щетініна Л. В., Касяненко Я. А. (2020). Digital HR – майбутнє кадрового адміністрування. Бізнес Інформ. № 1. С. 265–270. URL: https://doi.org/10.32983/2222-4459-2020-1- 265-270.
4. Ситник Й. С. (2017). Інтелектуалізація систем менеджменту підприємств: концепція, системний моніторинг та моделювання: монографія. Львів: Видавництво Львівської політехніки.
5. Захарчин Г. М., Поплавська Ж. В. (2017). Управління персоналом в контексті сучасних викликів. Актуальні проблеми економіки. № 4. С. 125–133. URL: http://nbuv.gov.ua/UJRN/ape_2017_4_15.
6. Інноваційні засади управління людськими ресурсами: можливості, виклики, пріоритети досягнення соціально-економічної безпеки: колективна монографія / за наук. ред. д-ра екон. наук, проф. Г. Ю. Міщук [Й. С. Ситник, О. Р. Сватюк]. Рівне: НУВГП, 2020. 408 с.
7. Цифровізація управлінської праці [Електронний ресурс] / О. Р. Сватюк, А. О. Захарец, Й. С. Ситник – Режим доступу до ресурсу: https://science.lpnu.ua/sites/default/files/journal-paper/2022/dec/29533/220972maket-214-226.pdf.
8. Onion Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://www.codeguru.com/csharp/understanding-onion-architecture/.
9. SOLID [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/SOLID.
10. Entity Framework Core [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/ef/.

11. CQRS [Електронний ресурс] – Режим доступу до ресурсу: https://www.eventstore.com/cqrs-pattern.
12. Middleware [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-7.0.
13. AWS S3 [Електронний ресурс] – Режим доступу до ресурсу: https://s3.console.aws.amazon.com/s3/home?region=us-east-1.
14. XUnit [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test.
15. Angular [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/docs.
16. AngularMaterial [Електронний ресурс] – Режим доступу до ресурсу: https://material.angular.io/.
17. Angular Testing [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/guide/testing-components-basics.
18. PeopleForce [Електронний ресурс] – Режим доступу до ресурсу: https://peopleforce.io/uk/request-demo.
19. Coloris HRM [Електронний ресурс] – Режим доступу до ресурсу: https://lms.coloris.com.ua/index.html.
20. Djinni [Електронний ресурс] – Режим доступу до ресурсу: https://djinni.co/my/dashboard/.
21. LinkedIn [Електронний ресурс] – Режим доступу до ресурсу: https://www.linkedin.com/.
22. GitHub [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/Soniabel/BoardingTracker.git.

# ДОДАТОК А

## Структурна діаграма бази даних