ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**Розробка HR Onboarding платформи з використанням .NET Core Web API, MS SQL, Angular**»

Виконав: студент 5 курсу, групи ППЗ–51
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Лаптєв А.О.
(прізвище та ініціали)
Керівник Золотухіна О. А.
(прізвище та ініціали)
Рецензент_____
(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти «Бакалавр»

Напрям підготовки - 121 - Інженерія програмного забезпечення"

ЗАТВЕРДЖУЮ

Завідувач кафедри

інженерії програмного забезпечення

О.В. Негоденко

«____» _____2023 року

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Лаптєв Андрій Олексійович

(прізвище, ім‘я, по батькові)

1.Тема роботи: «Розробка HR Onboarding платформи з використанням .NET Core Web API, MS SQL, Angular».

Керівник роботи Золотухіна Оксана Анатоліївна, к.т.н., доц., доцент кафедри ІПЗ,

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

затверджені наказом вищого навчального закладу від «24» лютого 2023р. №26

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

3. Вхідні дані до роботи:
   3.1 Науково-технічна література з питань, пов’язаних з розробкою HR технологій і веб-платформ
   3.2 Офіційна документація Microsoft SQL Server.
   3.3 Офіційна документація Angular.
   3.4 Офіційна документація .NET Core Web API.
   3.5 Стандарти і рекомендації з безпеки та захисту даних для веб-платформ.

4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).
   4.1 Аналіз актуальності та проблематики розроблюваної Onboarding платформи.
   4.2 Аналіз та вибір інструментів для реалізації Onboarding платформи.
   4.3 Проектування та розробка системи.
   4.4 Тестування програмного забезпечення.

5. Перелік демонстраційного матеріалу (назва основних слайдів)
   5.2 Мета, об’єкт, предмет дослідження.
   5.3 Задачі дипломної роботи
   5.4 Аналіз аналогів.
   5.5 Вимоги до програмного забезпечення
   5.6 Програмні засоби реалізації.
   5.7 Діаграма варіантів використання
   5.8 Схема бази даних
   5.9 Діаграма класів
   5.10 Екранні форми
   5.11 Апробація результатів дослідження.
   5.12 Висновки
6. Дата видачі завдання «25» лютого 2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	12.05.2023	Виконано
2	Аналіз та дослідження існуючих аналогів	14.05.2023	Виконано
3	Дослідження програмних засобів	15.05.2023	Виконано
4	Моделювання об'єкту проектування	11.05.2023	Виконано
5	Розробка функціоналу Onboarding платформи	12.05.2023	Виконано
6	Розробка обов'язкових демонстраційних матеріалів	18.05.2023	Виконано
7	Вступ, висновки, реферат	18.05.2023	Виконано
8	Попередній захист роботи	21.05.2023	Виконано
9	Здача роботи	01.06.2023	

Студент _____ Лаптєв А. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Золотухіна О. А.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 39 с., 1 табл., 14 рис., 12 джерел.

Об’єкт дослідження – процес HR Onboarding.

Предмет дослідження – програмне забезпечення процесу HR Onboarding.

Мета роботи – оптимізація процесу HR Onboarding за рахунок використання платформи, створеної засобами .NET Core, Web API, MS SQL і Angular.

В рамках даного дипломного проекту проведено дослідження поточних тенденцій у сфері HR Onboarding. Проаналізовано основні переваги і недоліки існуючих інструментів розробки та платформ. Висвітлено особливості розробки платформи HR Onboarding з використанням технологій .NET Core, Web API, MS SQL і Angular.

Платформа HR Onboarding була розроблена з використанням .NET Core та Web API для бекенду, MS SQL для управління базами даних, а Angular використовувався для створення фронтенду.

Дана платформа спрямована на поліпшення взаємодії між роботодавцями та новопризначеними співробітниками, забезпечуючи гладкий перехід до нового робочого місця. Практична значущість роботи полягає в створенні інтуїтивного, ефективного та адаптивного інтерфейсу для HR Onboarding; розробка бекенду, який може обробляти великі обсяги даних та інтегруватися з існуючими системами.

HR ONBOARDING, .NET CORE, WEB API, MS SQL, ANGULAR.

ЗМІСТ

Стор.

ВСТУП **8**

1 ОГЛЯД ІСНУЮЧИХ HR ONBOARDING ПЛАТФОРМ ТА АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ПРОЄКТУ **9**

1.1 Аналіз та загальна характеристика HR Onboarding платформ 9

1.2 Аналіз існуючих HR Onboarding систем 10

2 вИБІР ЗАСОБІВ рЕАЛІЗАЦІЇ **18**

2.1 Вибір бекенд технологій для створення додатку 18

2.2 Вибір фронтенд технологій для створення додатку 23

2.3 Обґрунтування вибору архітектури для API 25

2.4 Аналіз механізмів аутентифікації та авторизації 28

3. ПРОЕКТУВАННЯ І РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ **31**

3.1 Моделювання вимог 31

3.2 Дизайн інтерфейсу системи 33

3.3 Розробка діаграми класів 36

3.4 Розробка схеми бази даних 38

3.5 Розробка бекенд частини за допомогою .NET Core 42

3.6 Розробка фронтед частини за допомогою Angular 43

3.7 Unit тестування 44

ВИСНОВКИ **48**

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ **49**

ДОДАТОК А ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ **51**

ДОДАТОК Б ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ **57**

ВСТУП

У сучасному динамічному бізнесі важливим аспектом стає ефективне залучення та адаптація нових співробітників. Цей процес, зазвичай відомий як onboarding, стає все більш та більш цифровим, з використанням нових технологій та платформ для забезпечення плавного переходу нових працівників у компанію. Важливість цифрового HR onboarding стає очевидною, коли розглядається висока швидкість обороту персоналу.

Об’єктом дослідження є процес HR Onboarding.

Предметом дослідження є програмне забезпечення процесу HR Onboarding.

Мета роботи полягає в оптимізації процесу HR Onboarding за рахунок використання платформи, створеної засобами .NET Core, Web API, MS SQL і Angular.

Визначена мета досягається виконанням наступних задач:

– оцінка доступних систем HR Onboarding;

– дослідження технологій, які використовуються в HR Onboarding платформах;

– розробка технічного завдання для HR Onboarding платформи;

– розробка архітектури додатку;

– створення структури бази даних;

– розробка та тестування клієнтської частини платформи за допомогою Angular.

Було вибрано платформу для розробки (.NET Core), технологію для створення API (Web API), базу даних (MS SQL) та фреймворк для фронтенду (Angular). Крім того, було розроблено інтерфейс користувача, реалізовано основні функції платформи, після чого проведено тестування та внесено корективи на основі отриманих результатів.

1 ОГЛЯД ІСНУЮЧИХ HR ONBOARDING ПЛАТФОРМ ТА АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ПРОЄКТУ

1.1 Аналіз та загальна характеристика HR Onboarding платформ

HR Onboarding – це процес інтеграції нових співробітників у організаційну культуру та структуру компанії, який зазвичай включає навчання, адаптацію та соціалізацію. HR Onboarding платформи – це програми, розроблені для полегшення цього процесу.

В сучасному бізнесі такі платформи грають значущу роль у підвищенні ефективності цього процесу. Ці платформи є програмними рішеннями, які допомагають організувати, автоматизувати та вдосконалювати процеси онбордингу. Сучасні HR Onboarding платформи мають широкий функціонал, який включає, але не обмежується, такими рисами: автоматизоване заповнення форм, трекінг задач, навчання з використанням ігрових елементів (gamification), розробка індивідуальних планів адаптації та інше.

Сьогодні існує велика кількість HR Onboarding платформ, серед яких можна виділити такі, як Talmundo, Sapling, и Click Boarding. Кожна з цих платформ має свої сильні та слабкі сторони, але всі вони спрямовані на підвищення ефективності процесу онбордингу та забезпечення комфортного введення нових працівників в робочий процес. HR Onboarding платформи не лише підвищують ефективність процесу онбордингу, але й допомагають знизити витрати на нього. Завдяки автоматизації багатьох процесів, HR-спеціалісти можуть зосередитись на важливих задачах, замість того, щоб витрачати час на рутинні обов'язки.

При виборі HR Onboarding платформи слід враховувати специфіку компанії, її потреби та масштаби. Невеликі компанії, можливо, будуть задоволені менш складними рішеннями, тоді як великі корпорації з великою кількістю працівників і офісів у різних країнах потребуватимуть більш розгорнутих систем.

Враховуючи велику роль HR Onboarding платформ у сучасному бізнесі, слід очікувати, що їхній вплив та значимість лише зростатиме. Підтримка нових

технологій, таких як штучний інтелект і машинне навчання, можуть значно збільшити потенціал цих платформ, що дає можливість вдосконалити процеси онбордингу, зробити їх більш цікавими для нових працівників. Через ці системи вони можуть ознайомитися зі структурою компанії, її місією та цінностями, культурою, правилами та процедурами, а також важливими процесами та інструментами для роботи. Використання цих платформ допомагає новим працівникам швидше адаптуватися, а також почуватися більш впевнено та зрозуміло в новому робочому середовищі.

Водночас, платформи дають можливість HR-менеджерам ефективно відстежувати та аналізувати процес онбордингу. Вони можуть використовувати різні метрики та індикатори для оцінки ефективності онбордингу, включаючи задоволеність нових працівників, їхнє виконання та затримку на роботі. Ці дані можуть бути використані для постійного вдосконалення онбордингових процесів.

HR Onboarding платформи також можуть забезпечувати гнучкість та індивідуальність в процесі онбордингу. Через різні модулі та інструменти, HR-менеджери можуть створювати унікальні програми онбордингу, що враховують специфіку посади, відділу та індивідуальні потреби нового працівника.

Незважаючи на великі переваги цих платформ, вони також мають певні обмеження та труднощі. Зокрема, це включає потребу в їхній правильній інтеграції з іншими HR-системами та бізнес-процесами, високу вартість деяких рішень, а також труднощі з точки зору захисту даних та конфіденційності. Однак, враховуючи швидкий розвиток технологій та постійні інновації в цій галузі, можна очікувати, що ці труднощі будуть подолані в майбутньому.

1.2 Аналіз існуючих HR Onboarding систем

В цьому пункті наведено результати аналізу найвідоміших існуючих платформ для забезпечення процесу HR Onboarding.

По-перше, хотілось би відзначити платформу Talmundo (рис.1.1), яка протягом останнього десятиліття покращує досвіду роботи з новими

працівниками. Talmundo, заснована компанією Talentech у 2012 році, пропонує програмне забезпечення для управління персоналом, яке допомагає компаніям підвищити залученість співробітників, покращити продуктивність і забезпечити швидку адаптацію. Керівники та менеджери можуть використовувати цю платформу для залучення нових співробітників і підтримки взаємодії між ними.

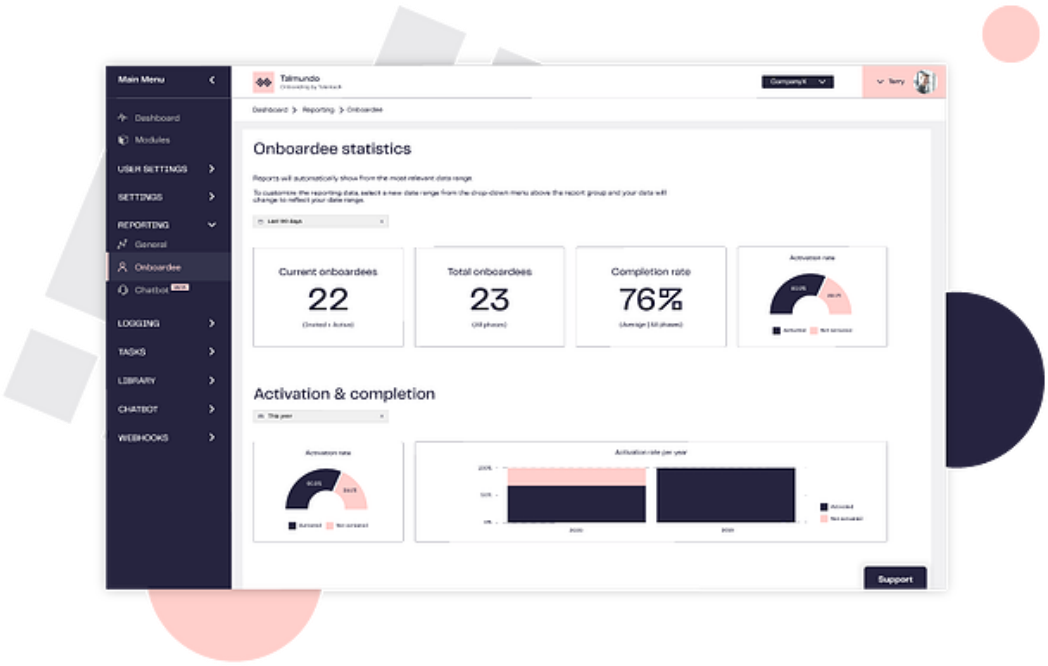


Рисунок 1.1 – Talmundo

Talmundo виділяється тим, що надає підтримку новим працівникам і скорочує час, потрібний їм, щоб стати продуктивними на своїх посадах. Ключові особливості Talmundo включають можливість для співробітників персоналізувати зміст навчання, дізнатися про цінності та культуру компанії, а також мати бачення свого майбутнього шляху з першого дня. Крім того, Talmundo пропонує шаблони процесу онбордингу, такі як контрольні списки та опитування, щоб спростити процес онбордингу.

Однією з помітних сильних сторін Talmundo є його вражаючі користувацькі форми, можливості керування документами та завданнями. Його основні функції охоплюють робочий процес адаптації зі списками справ, персоналізовані навчальні матеріали, гейміфіковані оцінки, міжорганізаційні звіти та цифровий

чат-бот для онбордингу. Talmundo також надає доступ в свою академію Onboarding Academy. Хоча Talmundo легко доступний з використанням різних пристроїв і операційних систем, наразі він не має мобільного додатка.

Talmundo обслуговує компанії будь-якого розміру, від малих (11-50 співробітників) до середніх (51-200 співробітників) і великих підприємств (201+ співробітників) у різних галузях, включаючи освіту та IT. Це дає змогу відділам кадрів, менеджерам і супервайзерам персоналізувати процес онбордингу, сприяючи інвестиціям і продуктивності.

Що стосується ціноутворення, Talmundo не розкриває свою модель ціноутворення на своєму веб-сайті. Зацікавлені користувачі повинні надіслати запит, щоб отримати інформацію про ціни, які визначаються на основі розміру компанії та бажаних послуг. Користувачам доступна безкоштовна пробна версія, щоб ознайомитися з функціями інструменту перед тим, як придбати ліцензію.

Для інтеграції з іншим програмним забезпеченням для управління персоналом Talmundo підтримує спеціальну інтеграцію API, що дозволяє компаніям підключати його до своїх існуючих систем, таких як SAP SuccessFactors або AFAS Software.

Talmundo надає підтримку клієнтам через різні канали, включаючи електронну пошту та чати підтримки. Платформа також пропонує секцію, де користувачі можуть знайти відповіді на поширені запитання. Багатомовна підтримка забезпечує доступність для користувачів з різних країн на мовах, яким вони віддають перевагу. [1]

Sapling (рис.1.2) - це потужна HR-платформа, яка забезпечує всеохоплюючий підхід до управління персоналом. Серед основних функцій Sapling — автоматизоване введення персоналу, керування даними про співробітників, трекінг їхніх відпусток і лікарняних, інтеграція з іншими бізнес-інструментами, такими як Slack, Google Suite та Salesforce.

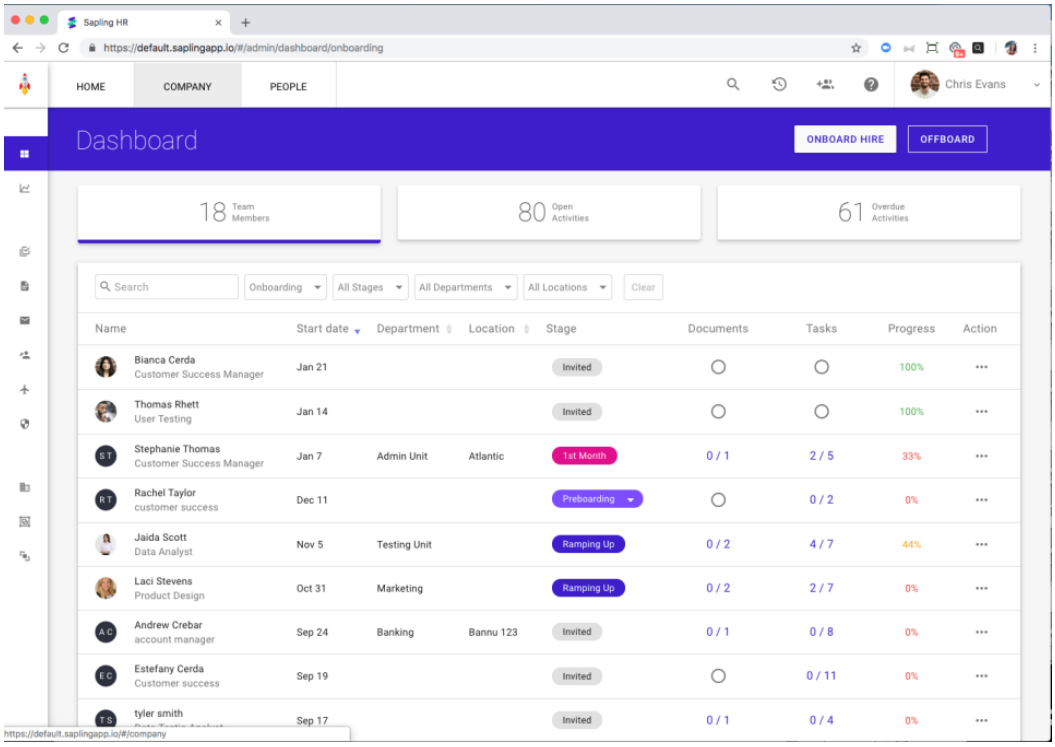


Рисунок 1.2 - Sapling

Плюси Sapling наведено нижче.

Інтуїтивний інтерфейс: платформа добре організована і проста у використанні, що знижує поріг входження для нових користувачів.

Гнучкість: Sapling надає можливість налаштування платформи під специфічні потреби вашого бізнесу.

Повна інтеграція: Sapling сумісний з багатьма популярними бізнес-інструментами, що значно покращує його функціональність.

Мінуси Sapling включають наступне.

Обмежений функціонал: хоча Sapling пропонує широкий спектр HR-інструментів, деякі з них можуть бути обмеженими в порівнянні з спеціалізованими рішеннями.

Ціна: Sapling може бути досить дорогим для малого або середнього бізнесу.

Щодо реалізації, Sapling легко впроваджується в будь-яку організацію, завдяки своїм автоматизованим процесам і гнучким налаштуванням. Все це робить Sapling відмінним вибором для компаній, що прагнуть оптимізувати управління персоналом.

Окрім основних можливостей, Sapling пропонує ряд додаткових функцій. Серед них — сучасні аналітичні інструменти для відстеження робочого часу, продуктивності та процесу навчання співробітників. Це дозволяє менеджерам отримувати детальний аналіз роботи команди та коригувати процеси відповідно до отриманих даних.

Однією з унікальних функцій Sapling є його система онбордингу. Sapling дозволяє автоматизувати процес введення нових співробітників, починаючи від першого дня роботи і до повного адаптування. Це не тільки економить час HR-менеджерів, але і забезпечує новачкам зручний і зрозумілий процес адаптації.

У зв'язку з розширенням можливостей дистанційної роботи, Sapling надає інструменти для ефективного управління віддаленими командами. Це включає в себе модулі для управління проектами, трекінгу робочого часу та комунікації між співробітниками.

Загалом, Sapling покликаний зробити процес управління персоналом якомога простішим і ефективнішим. Незважаючи на високу вартість і деякі обмеження, ця платформа стане незамінною для компаній, що прагнуть оптимізувати свої HR-процеси та підвищити загальну продуктивність. [3]

Click Boarding (рис.1.3) - це HR-платформа, яка орієнтована на поліпшення процесу ознайомлення нових працівників з організацією. Одним з основних плюсів Click Boarding є його спрощений, користувацький інтерфейс. Ця платформа є інтуїтивно зрозумілою для нових працівників, незалежно від їхнього рівня технічної підготовки. Крім того, Click Boarding надає роботодавцям зручні інструменти для слідкування за прогресом працівників впродовж процесу онбордингу.

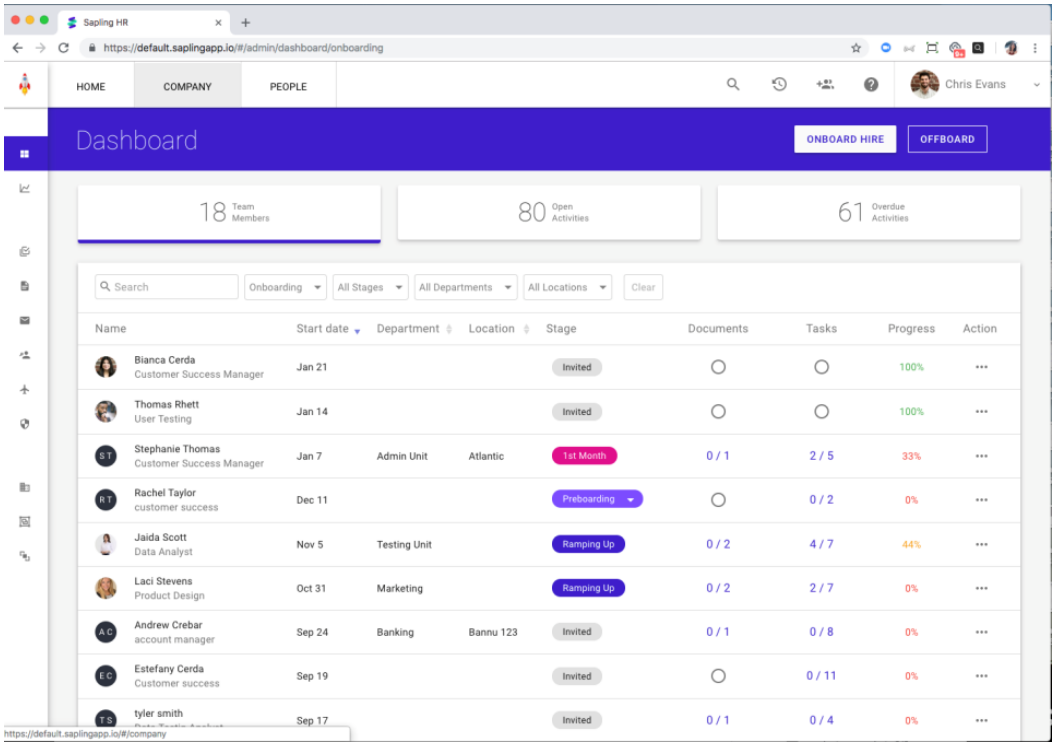


Рисунок 1.3 - Click Boarding

Другий важливий плюс - це гнучкість. Click Boarding підтримує не тільки онбординг, але і перехід, повернення з відпустки і зміни в роботі. Це означає, що платформа може бути використана для управління всіма видами взаємодій, включаючи працівників на повну ставку, часткову зайнятість, віддаленої роботи, контрактників, тимчасових працівників та інтернів.

Щодо мінусів, критичних недоліків важко знайти, але є декілька аспектів, які потребують уваги. Вартість Click Boarding може бути високою для деяких маленьких або середніх компаній, особливо якщо вони не планують часто використовувати платформу. Другий момент - потреба в оновленні. Click Boarding регулярно вносить оновлення для підтримки новітніх технологій, але користувачам потрібно переконатися, що вони мають останню версію.

Щодо реалізації, Click Boarding пропонує цілком цифровий процес, що полегшує онбординг і забезпечує відмінну прозорість. Роботодавці можуть налаштувати процес онбордингу так, щоб він відповідав їхнім специфічним потребам, і легко слідкувати за прогресом кожного працівника.

Додатково до згаданих вище плюсів, Click Boarding пропонує ряд інших переваг. По-перше, допомагає підвищити залученість працівників. Платформа використовує інтерактивні елементи та персоналізований контент, що спонукає працівників активно брати участь в процесі онбордингу.

По-друге, Click Boarding підтримує мультиязичність, що дозволяє організаціям легко впроваджувати працівників з різних країн та культурних контекстів.

Крім того, Click Boarding інтегрується з багатьма основними HR-системами, що робить його зручним інструментом для багатьох організацій.

Однак, варто зазначити, що, хоча Click Boarding пропонує багато корисних функцій, він може бути дещо перебільш складним для компаній, які не потребують такої великої гнучкості або персоналізації. В деяких випадках, менш складні або більш нішеві рішення можуть бути більш відповідними. [2]

Результати порівняльного аналізу HR Onboarding систем наведено в таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз HR Onboarding систем

Показник для аналізу	**Talmundo**	**Sapling**	**Click Boarding**
Переваги	Інтуїтивний інтерфейс, гнучкість, мультиязичність	Інтеграція з іншими HR системами, візуально привабливий інтерфейс	Швидке оформлення онбордингових процесів, мобільний додаток.
Ключові функції	Персоналізований онбординг, інтеграція з HR системами, звітність	Керування талантами, інтеграція з HR системами, звітність	Швидке оформлення онбордингових процесів, слідкування за прогресом, звітність

Продовження таблиці 1.1 – Порівняльний аналіз HR Onboarding систем

Показник для аналізу	**Talmundo**	**Sapling**	**Click Boarding**
Особливості реалізації	Хмарна модель, зручна для великих компаній	Хмарна модель, підходить для великих організацій	Хмарна модель, мобільний додаток
Недоліки	Немає мобільного додатку	Потребує велику кількість часу на впровадження	Обмеження в налаштуванні, відсутність мультиязичності

2 ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Вибір бекенд технологій для створення додатку

.NET Core Web API є оптимальним вибором для бекенд частини проекту з декількох причин.

Висока продуктивність: .NET Core, за офіційними даними, є однією з найшвидших технологій для веб-розробки на ринку. Особливо це важливо для великих систем, де кожна мілісекунда обробки запитів може відігравати значну роль.

Багатоплатформність: .NET Core підтримує розробку не тільки для Windows, але і для Linux та MacOS. Це дає змогу розгортати ваш проект на різних платформах без будь-яких змін в коді.

Безпечність: Microsoft постійно працює над забезпеченням .NET Core, тому ви можете бути впевнені, що ваша платформа захищена від більшості потенційних загроз.

Масштабованість: .NET Core дозволяє легко масштабувати ваш проект, що є особливо важливим для бізнес-орієнтованих додатків.

Підтримка Entity Framework: Entity Framework - це ORM (Object-Relational Mapper), який спрощує роботу з базами даних. І .NET Core відмінно працює з Entity Framework, дозволяючи легко взаємодіяти з базою даних.

Тестування: .NET Core має відмінну підтримку юніт тестування через XUnit, що дозволяє створювати надійні та стабільні додатки.

Інтеграція з Auth0: .NET Core дозволяє вам легко інтегруватись з різними сервісами автентифікації, такими як Auth0, що є важливим для забезпечення доступу до платформи. [4]

Далі потрібно визначитися з базою даних, яка використовуватиметься для проекту.

Microsoft SQL Server є відмінним вибором для багатьох проектів. Нижче наведено декілька причин, чому це виправдано:

Широка сумісність: Microsoft SQL Server взаємодіє з багатьма мовами програмування, включаючи .NET, Python, Java, R, Ruby, PHP та інші. У нашому випадку використання .NET Core Web API спрощує інтеграцію з MS SQL Server.

Надійність та безпека: Microsoft SQL Server відомий своєю високою надійністю і безпекою. Це важливо для захисту даних працівників в HR-системі.

Широкі можливості для розширення: Microsoft SQL Server пропонує гнучкі можливості для масштабування та оптимізації, включаючи підтримку розподілених баз даних та кластерів.

Інтеграція з Entity Framework: Entity Framework є одним з найпопулярніших ORM для .NET і він працює чудово з MS SQL Server. Це робить роботу з базою даних набагато легшою і ефективнішою, дозволяючи вам працювати з об'єктами в коді замість SQL-запитів.

Підтримка Transact-SQL (T-SQL): T-SQL - це діалект SQL, що пропонується Microsoft. Він надає багато корисних функцій, яких немає в стандартному SQL, що дозволяє розробникам створювати більш складні та гнучкі запити.

Безпечне зберігання даних: MS SQL Server пропонує можливості шифрування на рівні бази даних, що є важливим для зберігання чутливої інформації про працівників.

Далі перейдемо до ORM фреймворку, оскільки це дуже важливо під час роботи з базою даних.

Entity Framework (EF) є потужною технологією, яка може значно спростити роботу з базами даних у нашому проекті. Вибір Entity Framework для проекту є обґрунтованим з наступних причин:

Інтеграція з .NET: Entity Framework є частиною екосистеми .NET, що забезпечує плавну інтеграцію з іншими частинами вашого стека технологій, включаючи .NET Core Web API.

Code-First Підхід: Entity Framework дозволяє вам використовувати code-first підхід, який включає створення моделей в коді, а потім генерування бази даних на основі цих моделей. Це може полегшити роботу, особливо при розробці прототипів або при змінах в структурі даних.

LINQ-запити: Entity Framework дозволяє використовувати Language Integrated Query (LINQ) для роботи з даними, що робить код більш читабельним та легким для розуміння.

Щільність коду: Завдяки Entity Framework, вам не потрібно писати велику кількість SQL-коду. Замість цього, ви можете працювати з більш абстрактними конструкціями на рівні об'єктів.

Автоматичне відстеження змін: Entity Framework відстежує зміни, які ви робите в ваших об'єктах, і автоматично генерує відповідний SQL-код для внесення цих змін в базу даних.

Підтримка транзакцій: Entity Framework підтримує транзакції на рівні бізнес-логіки, що дозволяє гарантувати консистентність даних незалежно від складності операцій.

Міграції: EF дозволяє легко створювати та управляти міграціями бази даних, що є особливо корисним при впровадженні змін в моделі даних.

Незалежність від СУБД: EF є ORM, що підтримує різні СУБД. Хоча ви використовуєте MS SQL Server, ви могли б легко перейти на іншу базу даних з мінімальними змінами в коді, якщо коли-небудь виникне така потреба.

Так само дуже важливим аспектом розробки є Unit-тестування, для цього вибір припав на фреймворк під назвою XUnit

XUnit є відмінним вибором для юніт-тестування в .NET проектах з наступних причин.

Підтримка Сучасних Функцій та Конвенцій: XUnit розроблено з урахуванням сучасних практик тестування і підтримує широкий спектр функцій, таких як data-driven тестування, теорії, inline data, та custom test attributes. Ці функції дозволяють писати більш гнучкі тести і легше адаптуватися до змін у вимогах до програмного забезпечення.

Ізоляція Тестів: XUnit природним чином ізолює тести, виконуючи кожен тест в окремому класі. Це означає, що стани тестів не перетинаються, що сприяє точності тестів та робить їх надійнішими.

Контрольний Порядок Виконання Тестів: XUnit не визначає конкретний порядок виконання тестів, що спонукає розробників до створення незалежних тестів. Це дуже корисна практика, яка забезпечує надійність тестового покриття.

Гнучкість і Масштабованість: XUnit може легко впоратися з великою кількістю тестів і має можливість паралельного виконання тестів, що забезпечує високу продуктивність та ефективність тестування.

Інтеграція з .NET: Оскільки XUnit створений спеціально для .NET, він інтегрується гладко з іншими технологіями .NET, такими як MS SQL Server, Entity Framework та .NET Core Web API.

У випадку нашого проекту, використання XUnit дозволить вам розробити добре структуровані, надійні тести, які забезпечать високу якість коду та упевненість у його правильній роботі.

Для управління користувачами буде використовуватися сервіс Auth0.

Auth0 є одним із провідних рішень на ринку в області Identity as a Service (IDaaS). Використання Auth0 для керування користувачами має ряд переваг:

Спрощена аутентифікація: Auth0 надає готові до використання рішення для різних сценаріїв аутентифікації, що дозволяє розробникам не витрачати час на створення таких систем вручну.

Багатий набір можливостей: Auth0 підтримує широкий спектр протоколів ідентифікації, включаючи OpenID Connect, OAuth 2.0 та SAML. Також сервіс пропонує можливості для соціальної автентифікації через численні платформи, такі як Google, Facebook, LinkedIn тощо.

Безпека: Auth0 забезпечує надійну безпеку, використовуючи сучасні стандарти і практики. Він також надає автоматичне шифрування паролів та можливість додавати двофакторну аутентифікацію.

Масштабування: Як хмарне рішення, Auth0 легко масштабується для великих об'ємів користувачів, забезпечуючи надійне керування ідентифікаторами, незалежно від об'єму користувачів.

Інтеграція: Auth0 може бути легко інтегрований з багатьма додатками та платформами, включаючи .NET Core Web API, що використовується в нашому проекті.

Гнучкість: Auth0 надає розробникам гнучкість у створенні та кастомізації воркфлоу аутентифікації, що відповідає потребам їх додатків.

Отже, використання Auth0 у нашому проекті дозволяє зосередитись на основних аспектах HR Onboarding платформи, забезпечуючи в той же час ефективне, безпечне та гнучке керування користувачами.

Останній бекенд аспект, який дуже важливо виділити - це документація. Для цих задач було обрано Swagger.

Документація API: Swagger надає зручний спосіб створити і підтримувати документацію для нашого API. Ви можете визначити всі ендпоінти, їх параметри та типи даних, а також описати різні можливості API. Це допоможе вашій команді та розробникам зрозуміти, як використовувати ваш API, і спростить їм процес інтеграції з ним.

Тестування API: Swagger надає вбудовану можливість для тестування API. Ви можете виконувати запити до вашого API, передавати необхідні параметри і перевіряти результати. Це дозволяє швидко перевірити, чи працює ваш API правильно, і забезпечити його якість.

Генерація клієнтського коду: Swagger може згенерувати автоматично клієнтський код API для різних платформ і мов програмування. Це заощадить час та зусилля вашої команди, оскільки вони зможуть швидко створити клієнтську частину, використовуючи згенерований код, і не потребуватимуть ручного написання всіх необхідних запитів.

Інтерактивна документація: Swagger створює інтерактивну документацію для API, що дозволяє розробникам випробувати ваші ендпоінти прямо в браузері. Вони можуть виконувати запити, переглядати результати і навіть спробувати різні комбінації параметрів, щоб побачити, як API реагує. Це допомагає зрозуміти функціональні можливості вашого API і прискорює процес розробки. [11]

Інтеграція з іншими інструментами: Swagger може бути легко інтегрований з іншими інструментами розробки, такими як Postman або автоматичні тестувальні інструменти. Ви можете імпортувати ваше специфікацію Swagger до цих інструментів і використовувати їх для подальшого тестування, автоматичної генерації документації або створення колекцій запитів.

Загалом, використання Swagger дозволяє створити документацію, тестувати, генерувати клієнтський код та спрощує інтеграцію API з іншими інструментами.

2.2 Вибір фронтенд технологій для створення додатку

Даний пункт містить обґрунтування вибору фронтенд фреймворків та бібліотек проєкту. Angular є потужним фреймворком для розробки веб-додатків, який надає розробникам зручні інструменти та широкі можливості. Далі наведено переваги вибору Angular.

Односторінкові додатки: Angular є одним з найкращих фреймворків для створення односторінкових веб-додатків (SPA). Він надає широкий набір інструментів та можливостей, що спрощують процес створення цих додатків.

TypeScript: Angular використовує TypeScript, що є строго типізованою мовою, що може полегшити процес розробки, виявлення помилок на ранніх стадіях і покращити читабельність коду.

Модульність: фреймворк базується на модулярному підході до розробки, що сприяє кращій організації коду і підтримці.

Двосторонній Data Binding: Angular надає потужну підтримку двостороннього зв'язування даних, що сприяє автоматичному оновленню представлення, коли модель змінюється, і навпаки.

Вбудовані сервіси: фреймворк надає ряд вбудованих сервісів, таких як HTTP клієнт для роботи з API, роутинг, форми та інше.

Тестування: Angular розроблений з урахуванням тестування. Він включає засоби для unit-тестування компонентів, сервісів, директив та інших частин додатка.

Інтеграція: Angular добре інтегрується з різними бібліотеками та фреймворками для створення інтерфейсу, такими як Angular Material, Bootstrap і Tailwind, що забезпечує гнучкість в дизайні.

Спільнота та документація: фреймворк має велику активну спільноту та відмінну документацію, яка може бути великим допоміжним ресурсом при розробці.

Dependency Injection: Angular використовує систему впровадження залежностей, що дозволяє легко замінювати компоненти та збільшує гнучкість та масштабованість вашого додатка.

Підтримка Mobile: за допомогою інструментів, таких як Ionic, Angular дозволяє розробляти мобільні додатки, що використовують одну і ту ж базу коду.

Для легкої та швидкої реалізації необхідного інтерфейсу було вибрано відповідні допоміжні UI бібліотеки Angular Material, які мають нижченаведені переваги.

Готовий дизайн та стилізація: Angular Material надає широкий набір готових компонентів та стилів, які відповідають Material Design - сучасному та естетичному дизайну, рекомендованому Google. Це означає, що можна швидко та легко створити привабливий та єдиноформний вигляд для платформи, не витрачаючи багато часу на дизайн та стилізацію.

Компонентна архітектура: Angular Material пропонує великий вибір готових компонентів, таких як кнопки, форми, таблиці, каруселі, спливаючі вікна та багато інших. Ці компоненти легко використовувати та налаштовувати, що дозволяє вам швидко розробляти функціональні елементи інтерфейсу користувача. Крім того, використання однорідної компонентної архітектури полегшує управління та підтримку коду.

Адаптивний та респонсивний дизайн: бібліотека надає можливість легко створювати адаптивні та респонсивні інтерфейси користувача. Це означає, що

платформа буде добре працювати на різних пристроях та екранах розміром, включаючи комп'ютери, планшети та смартфони. Можна використовувати готові медіа-запити та адаптивні компоненти для розміщення та організації контенту в залежності від розміру екрана користувача.

Легкість інтеграції: Angular Material розроблено спеціально для Angular, тому воно ідеально поєднується з цим фреймворком. Інтеграція Angular Material з вашим Angular додатком проста і зручна, що дозволить вам швидко розпочати роботу з готовими компонентами та стилями, забезпечуючи при цьому оптимальну продуктивність.

Документація та підтримка: бібліотека має детальну та зрозумілу документацію, яка надає необхідні вказівки та приклади для роботи з компонентами. Крім того, спільнота Angular є досить великою і активною, тому ви можете знайти багато ресурсів та підтримки, які допоможуть вам у вирішенні проблем та вдосконаленні платформи.

Загалом, використання Angular Material дозволяє зосередитися на розробці функціоналу HR Onboarding платформи, не витрачаючи багато часу на реалізацію дизайну та стилізації.

2.3 Обґрунтування вибору архітектури для API

Дуже важливою частиною розробки API для додатків є правильно побудована архітектура з можливістю розширення функціоналу і подальшою його підтримкою. В роботі було обрано архітектурний підхід Onion Architecture (рис.2.1).

Onion Architecture, відома також як Hexagonal або Ports and Adapters, є надзвичайно корисною для створення гнучких і відновлюваних систем. Особливо вона корисна для додатків з API, оскільки ця архітектура здатна розділити логіку і залежності, спростивши таким чином розробку і підтримку коду.

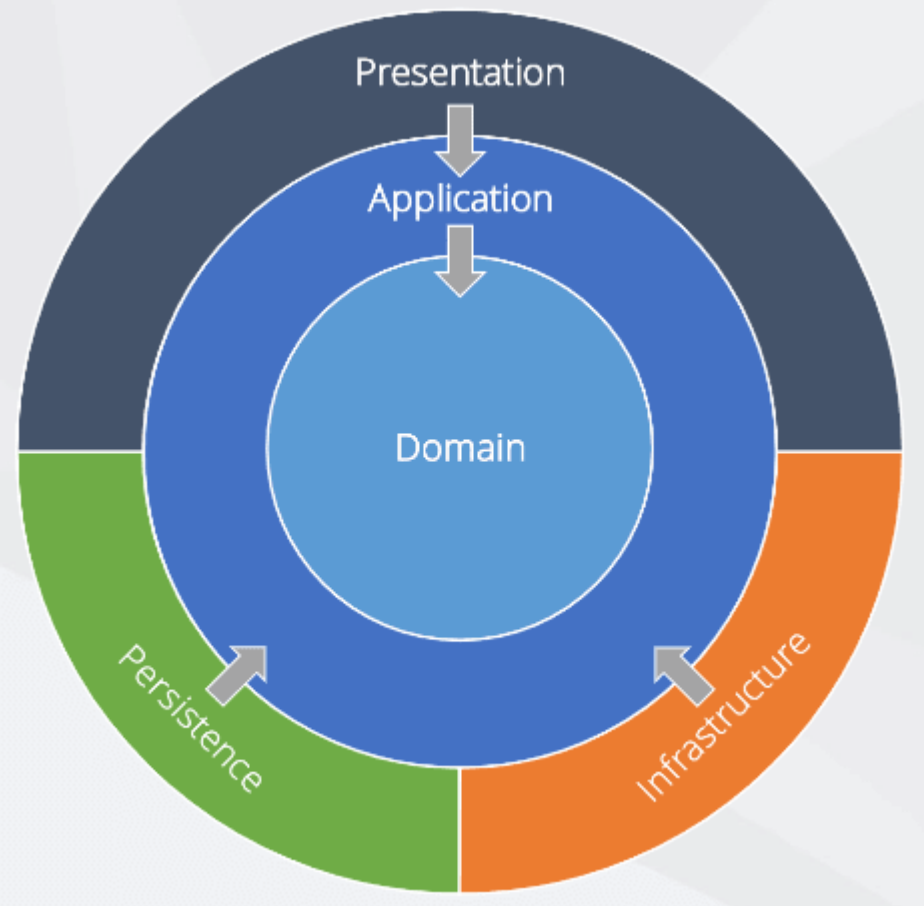


Рисунок 2.1 - Onion Architecture

Переваги Onion Architecture.

Адаптивність до змін: оскільки вона заснована на принципі залежностей, Onion Architecture робить ваш код гнучким і легким для адаптації до змін.

Тестування: дозволяє легше створювати модульні тести, оскільки код не залежить від інфраструктури або зовнішніх залежностей.

Підтримка коду: за допомогою Onion Architecture, код можна легко підтримувати і розвивати, оскільки ви можете змінювати частини системи без впливу на інші.

Структура Onion Architecture.

Основна ідея Onion Architecture полягає в тому, що зовнішні шари залежать від внутрішніх, а не навпаки.

Доменний шар: цей шар є найважливішим у Onion Architecture. Він містить всю бізнес-логіку і бізнес-правила.

Шар сервісів: цей шар є місцем для оркестрації бізнес-операцій і взаємодії з інфраструктурою.

Інтерфейси (API): цей шар містить всю логіку представлення і може включати в себе REST API, GraphQL API або будь-який інший тип API.

Інфраструктурний шар: цей шар містить всі деталі, пов'язані з інфраструктурою, такі як доступ до баз даних, файлових систем, мережі тощо.

Onion Architecture є великим вибором для більшості проектів API, особливо для тих, які очікують великих змін в майбутньому або потребують високої масштабованості.

Висока гнучкість: структура шарів Onion Architecture дозволяє з легкістю внести зміни у визначений шар без впливу на інші. Завдяки цьому, можна модернізувати систему, наприклад, оновлюючи базу даних або змінюючи логіку бізнес-правил, не переписуючи весь код.

Підтримка різних типів інтерфейсів: одна з особливостей Onion Architecture полягає в тому, що вона не залежить від конкретного типу інтерфейсу. Це означає, що ви можете використовувати різні типи API (REST, GraphQL тощо) в одному проекті без будь-яких проблем з сумісністю.

Стійкість до помилок: Onion Architecture підтримує високу стійкість до помилок, оскільки код, який містить бізнес-логіку, відділений від інфраструктури. Це означає, що навіть якщо щось піде не так з зовнішнім сервісом або базою даних, це не призведе до повної втрати функціональності додатку.

Модульність: через віддільність логіки і інфраструктури, Onion Architecture сприяє модульності. Ви можете легко додавати, замінювати або видаляти модулі, не впливаючи на загальну структуру програми.

Простота масштабування: завдяки своїй гнучкості і модульності, Onion Architecture дозволяє легко масштабувати систему. Це може бути корисно, наприклад, коли кількість користувачів або обсяг даних росте. [10]

У загальному підсумку, Onion Architecture є міцним фундаментом для будь-якого проекту API.

2.4 Аналіз механізмів аутентифікації та авторизації

В системі використовується модель аутентифікації на основі Auth0 flow (рис.2.2), яка є частиною додатку розробленого на Angular для фронтенду та .NET Core API для бекенду. Ця модель аутентифікації використовує JWT (JSON Web Token) токени для визначення ідентифікаційних даних користувача.

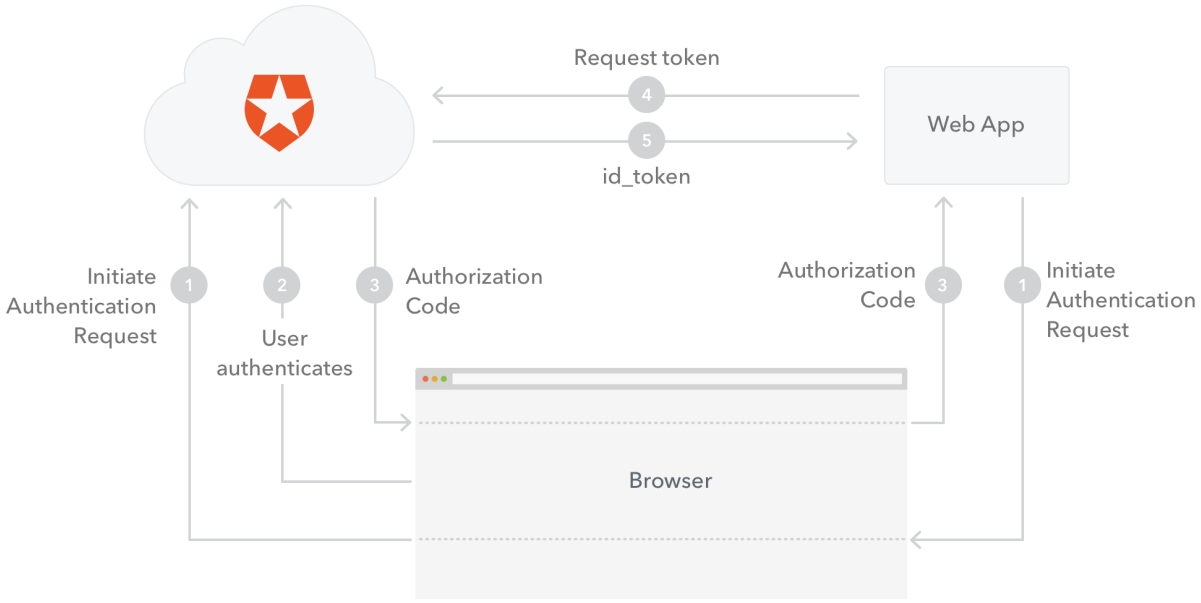


Рисунок 2.2 - Auth0 flow

Розглянемо обробку дій користувача, який намагається виконати вхід до системи. Користувач вводить свої облікові дані на веб-сайті, розробленому на Angular. Auth0 є постачальником перевірки справжності та служби ідентифікації. Після введення облікових даних, Auth0 проводить процес аутентифікації, перевіряючи введені користувачем дані.

Після успішної аутентифікації, Auth0 генерує JWT, який є безпечним способом представлення набору інформації між двома сторонами. JWT, який генерується Auth0, містить в собі унікальний ідентифікатор користувача, а також інформацію про поточну сесію, яка прив'язана до Auth0 додатку.

JWT тоді відправляється назад на клієнтську сторону і зберігається в локальному сховищі. В подальшому, при виконанні будь-якого запиту до сервера, цей токен додається в заголовки запиту.

.NET Core API, в свою чергу, використовує цей токен для перевірки справжності та авторизації користувача. При отриманні запиту, .NET Core API використовує токен для встановлення ідентичності користувача, здійснюючи зворотний запит до Auth0 для перевірки справжності токена.

Отже, суть цієї моделі полягає у тому, що Auth0 виконує всю важку роботу по аутентифікації та створенню JWT, в той час як .NET Core API використовує цей токен для визначення ідентичності користувача. Це розподіляє відповідальність і забезпечує високий рівень безпеки, оскільки жодні облікові дані користувача не передаються між сервером та клієнтом після первісної аутентифікації.

Особливістю даної моделі є те, що вся аутентифікаційна інформація зберігається в токені, що унеможливлює несанкціонований доступ до системи без наявності відповідного токена. Це забезпечує додатковий рівень захисту від потенційних зловмисних дій.

Auth0 служить надійною платформою для аутентифікації, оскільки вона включає різноманітні заходи безпеки, такі як двофакторна аутентифікація, блокування на основі IP, автоматичне блокування після певної кількості невдалих спроб входу та інше. Використовуючи Auth0, ми можемо концентруватися на розробці основної частини нашого додатку, маючи впевненість, що процес аутентифікації в нашій системі є безпечним та надійним.

Додатково, Auth0 надає гнучкість у виборі методу аутентифікації. Це може бути стандартна форма входу з ім'ям користувача та паролем, соціальна аутентифікація (Facebook, Google тощо), або корпоративна аутентифікація (LDAP, SAML, AD тощо). Така гнучкість дозволяє використовувати Auth0 в різних сценаріях, в залежності від потреб конкретного додатку. [6]

У кінцевому підсумку, модель аутентифікації на основі Auth0, яка використовується в нашій системі, дозволяє забезпечити надійну безпеку, здатність масштабувати та гнучкість, необхідну для сучасних веб-додатків.

3. ПРОЕКТУВАННЯ І РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Моделювання вимог

Моделювання вимог для платформи виконано за допомогою діаграми варіантів використання (рис.3.1). Ця діаграма є важливим інструментом для аналізу та визначення можливих шляхів використання системи чи програмного забезпечення. Залежно від ваших вимог можна використовувати ці дані різними способами. Нижче наведено кілька способів їх використання.

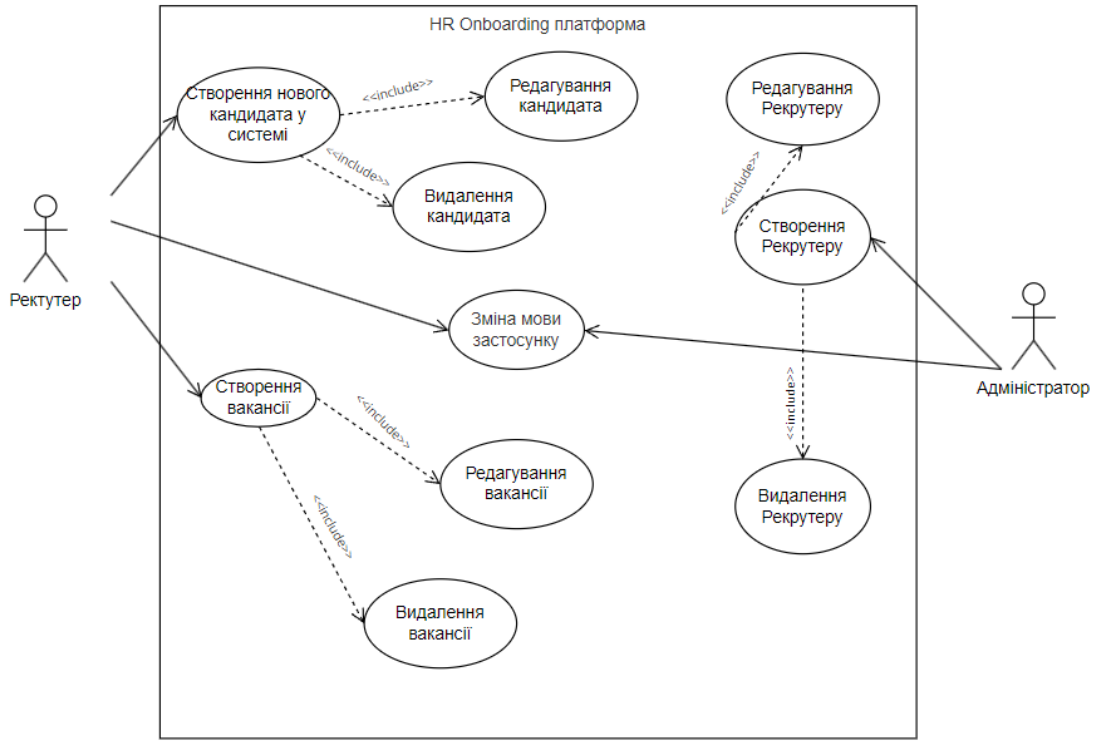


Рисунок 3.1 - Діаграма варіантів використання

Визначення функціональних можливостей та взаємодії акторів з ними - основна мета діаграм варіантів використання. Отримання загального уявлення про систему особливо корисно при презентації менеджерам або зацікавленим сторонам. Можна виділити акторів, що взаємодіють з системою, та функціональні

можливості, які надає система, не занурюючись в деталі внутрішньої роботи системи. [9]

Акторами HR Onboarding платформи є Рекрутер та Адміністратор.

Рекрутер має ряд основних функцій, які надають йому можливість керувати процесом набору персоналу. Опишемо прецеденти актора Рекрутер

Прецедент «Створення кандидата» описується наступним сценарієм:

1. Рекрутер вводить необхідну інформацію про кандидата в систему.
2. Система перевіряє введені дані на відповідність формату.
3. Система створює новий профіль кандидата.

Прецедент «Редагування кандидата» описується наступним сценарієм:

1. Рекрутер вибирає профіль кандидата для редагування.
2. Рекрутер вносить зміни в інформацію про кандидата.
3. Система зберігає зміни.

Прецедент «Видалення кандидата» описується наступним сценарієм:

1. Рекрутер вибирає профіль кандидата для видалення.
2. Система видаляє вибраний профіль кандидата.

Прецедент «Створення вакансії» описується наступним сценарієм:

1. Рекрутер вводить інформацію про вакансію в систему.
2. Система перевіряє введені дані на відповідність формату.
3. Система створює нову вакансію.

Прецедент «Редагування вакансії» описується наступним сценарієм:

1. Рекрутер вибирає вакансію для редагування.
2. Рекрутер вносить зміни в інформацію про вакансію.
3. Система зберігає зміни.

Прецедент «Видалення вакансії» описується наступним сценарієм:

1. Рекрутер вибирає вакансію для видалення.
2. Система видаляє вибрану вакансію.

Прецедент «Зміна мови» описується наступним сценарієм:

1. Рекрутер вибирає потрібну мову в налаштуваннях.
2. Система змінює мову інтерфейсу.

Далі наведено опис прецедентів актора Адміністратор.

Прецедент «Створення рекрутеру» описується наступним сценарієм:

1. Адміністратор вводить необхідну інформацію про рекрутера в систему.
2. Система перевіряє введені дані на відповідність формату.
3. Система створює новий профіль рекрутера.

Прецедент «Редагування рекрутеру» описується наступним сценарієм:

1. Адміністратор вибирає профіль рекрутера для редагування.
2. Адміністратор вносить зміни в інформацію про рекрутера.
3. Система зберігає зміни.

Прецедент «Видалення рекрутеру» описується наступним сценарієм:

1. Адміністратор вибирає профіль рекрутера для видалення.
2. Система видаляє вибраний профіль рекрутера.

Прецедент «Зміна мови» описується наступним сценарієм:

1. Адміністратор вибирає потрібну мову в налаштуваннях.
2. Система змінює мову інтерфейсу.

3.2 Дизайн інтерфейсу системи

Зазначені функціональні можливості користувачів реалізовано у вигляді інтерфейсних форм, наведених далі.

Екранну форму для перегляду популярних рекрутерів і вакансій на головній сторінці представлено на рис 3.2. На сторінці розміщено два основні блоки із закругленими кутами. Кожен блок має список підблоків з іменами та фітографіями вакансій та рекрутерів відповідно.

Екранну форму для перегляду списку кандидатів представлено на рис.3.3, вона включає в себе список кандидатів. Кожен блок з кандидатів містить наступну інформацію: повне ім'я, адреса електронної пошти, номер телефону, опис. Також доступні кнопки редагування та видалення.

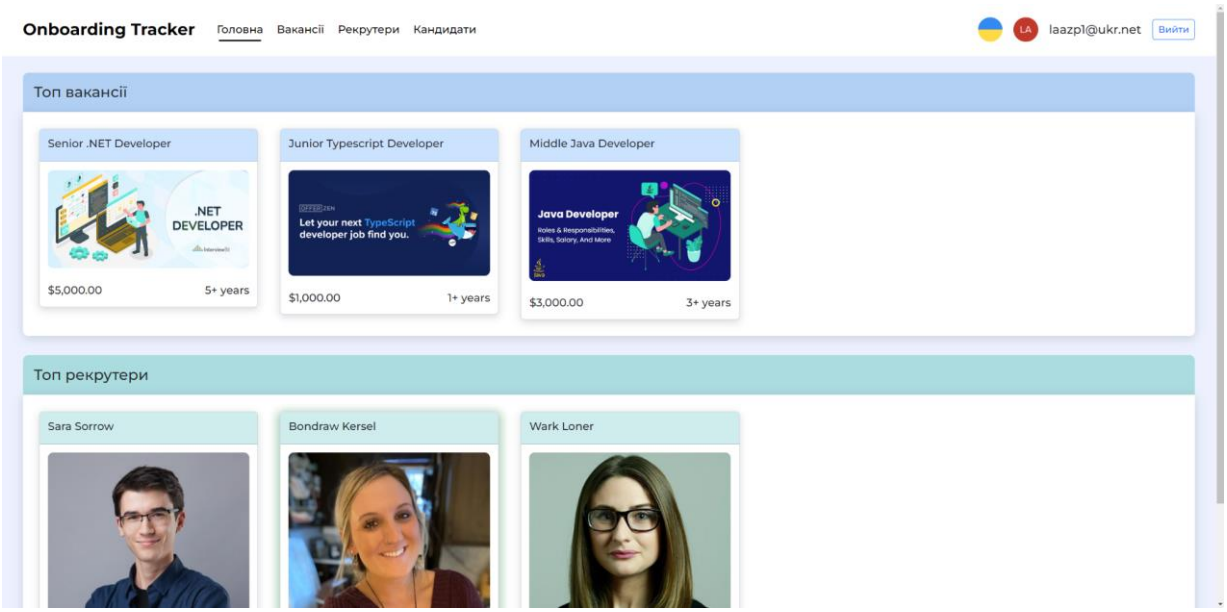


Рисунок 3.2 - Головна сторінка

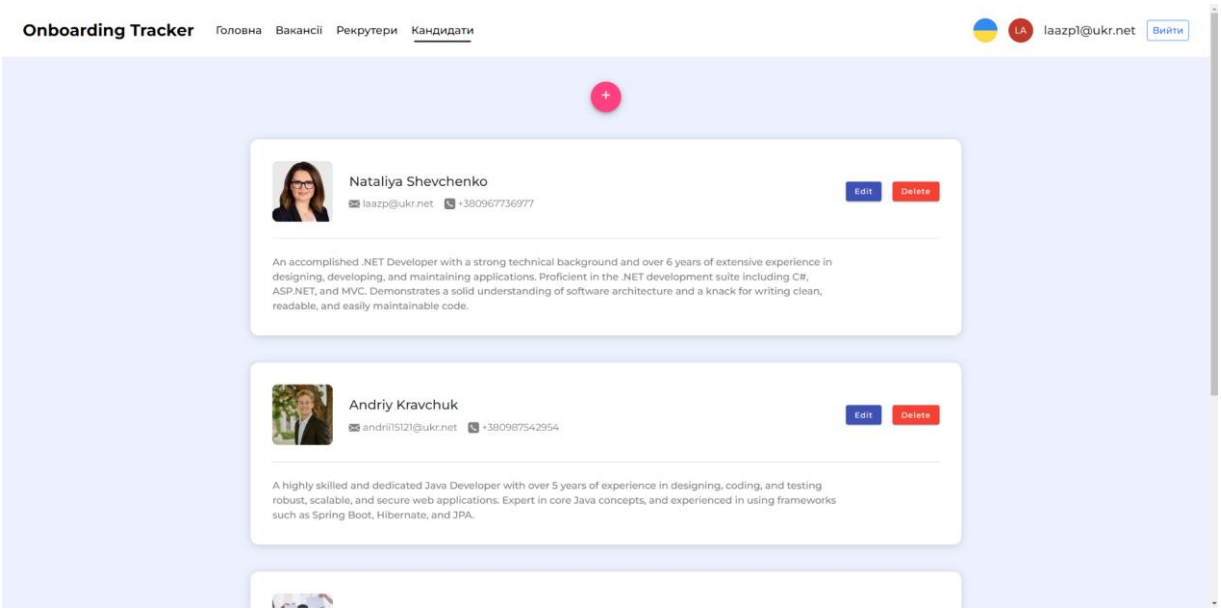


Рисунок 3.3 - Кандидати

Редагування та створення кандидатів представлено екранною формою на рис.3.4, яка включає список полів для редагування інформації про кандидата. Це - ім'я, прізвище, адреса електронної пошти, досвід, номер телефону, посилання на фотографію, рівень кваліфікації, опис та вміння кандидата.

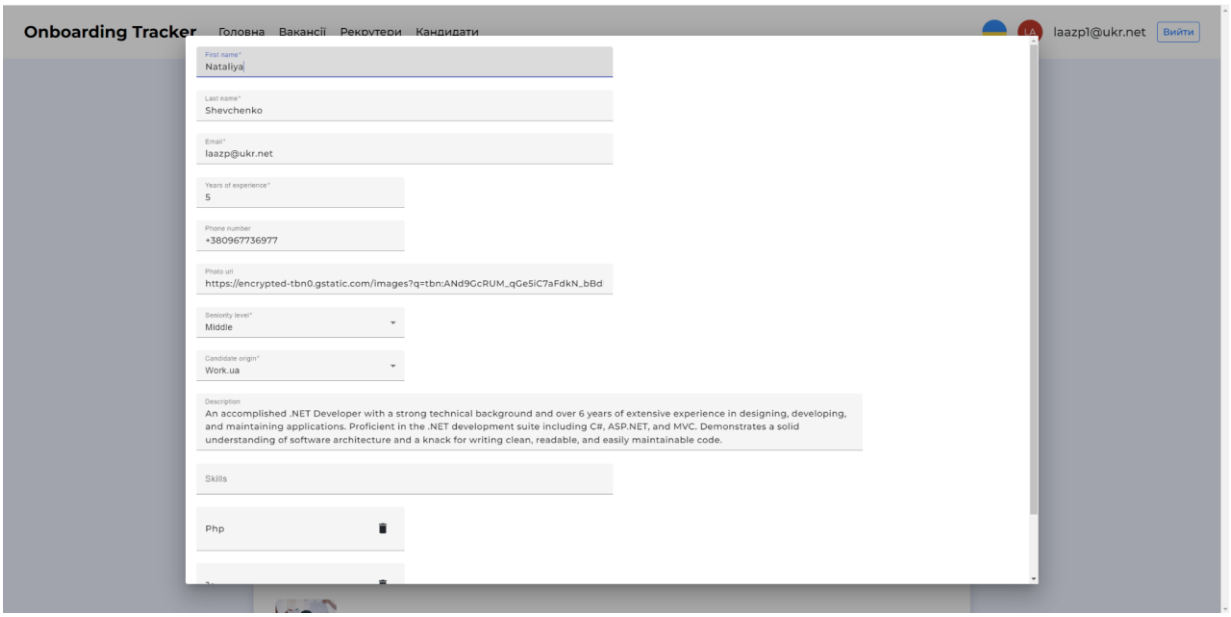


Рисунок 3.4 - Редагування або створення кандидату

Далі розглянемо інтерфейс адміністратора, який має можливість переглядати (рис. 3.5), створювати та редагувати (3.6) нових рекрутерів.

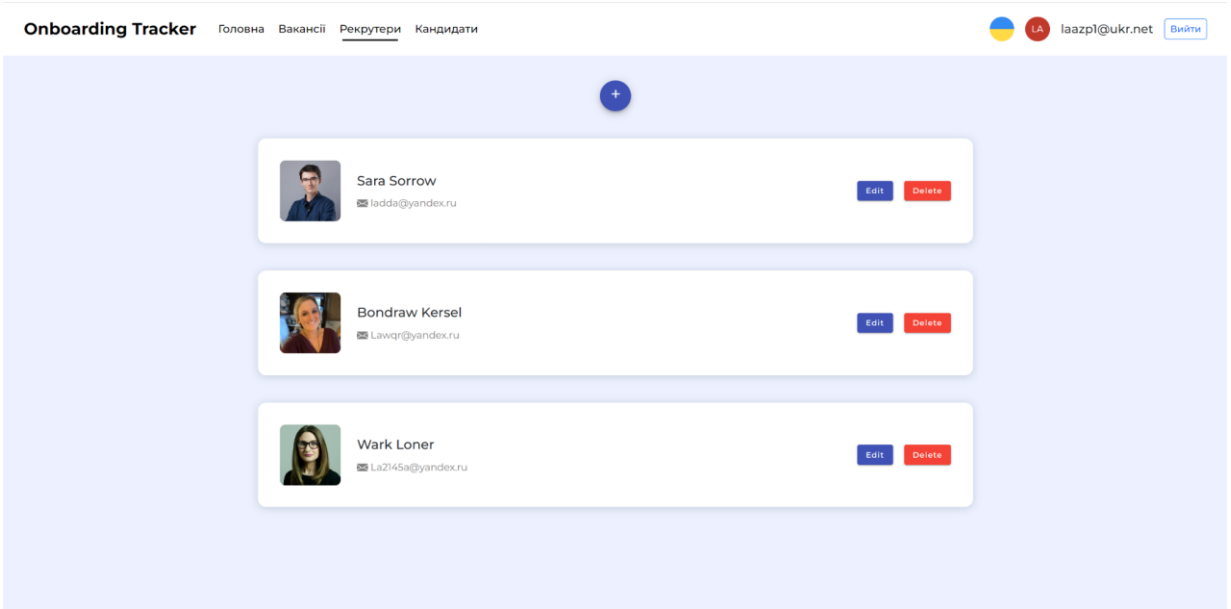


Рисунок 3.5 - Рекрутери

На сторінці зі списком рекрутерів кожен блок надає таку інформацію: повне ім'я рекрутера, адреса електронної пошти. Також доступна кнопка редагування та видалення. При натисканні на кнопку редагування ми потрапляємо на екранну

форму з наступними полями: ім'я, прізвище, адреса електронної пошти, посилання на фотографію.

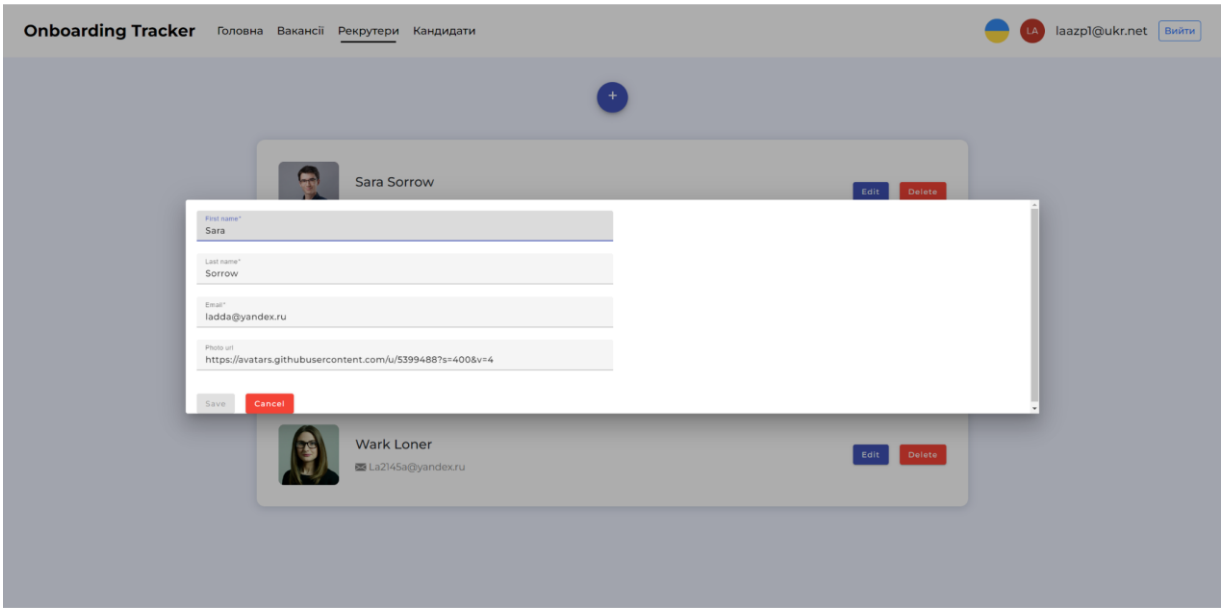


Рисунок 3.6 - Редагування або створення рекрутеру

Окрім цього, є можливість змінити мову інтерфейсу на українську або англійську. Цей функціонал доступний всім користувачам, що дозволяє використовувати додаток у зручному для них форматі.

3.3 Розробка діаграми класів

Діаграма класів (рис.3.7) - це важлива частина Unified Modeling Language (UML), яка використовується для візуалізації структури системи, представляючи її основні елементи, такі як класи, атрибути, операції та взаємодії. Завдяки своїй здатності чітко відображати відносини між різними компонентами, діаграма класів стала незамінним інструментом у руках розробників, архітекторів та бізнес-аналітиків.

У контексті .NET Core API, діаграма класів може допомогти у відображенні його основних компонентів і взаємодій. Наприклад, класи можуть представляти різні сутності (наприклад, користувачі, товари, транзакції тощо), атрибути -

властивості цих сутностей, а операції - дії, які можна виконати з цими сутностями (наприклад, додавання нового користувача, видалення товару тощо). Відносини між класами можуть представляти зв'язки між різними сутностями (наприклад, користувач може мати одну або кілька транзакцій). Використовуючи діаграму класів, можна краще розуміти структуру API, а також побачити, як різні компоненти взаємодіють між собою.

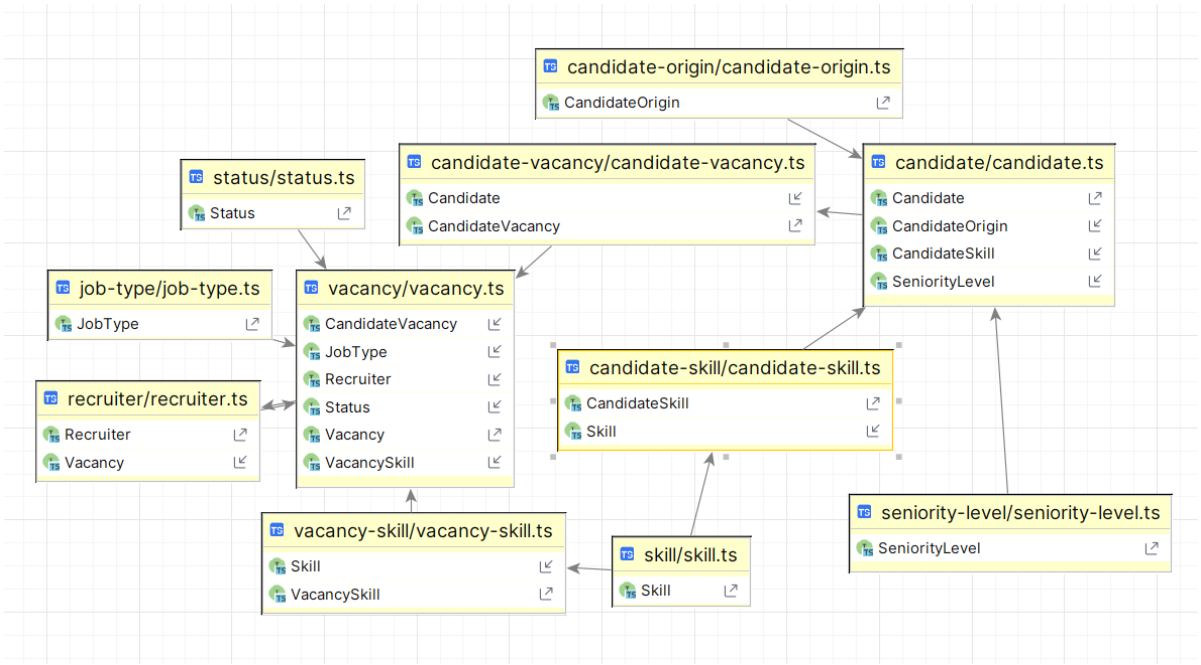


Рисунок 3.7 - Діаграма класів

Клас "Candidate" використовується для представлення кандидата, що шукає роботу. Кожен кандидат має унікальний ідентифікатор "Id", ім'я "FirstName", прізвище "LastName", кількість років досвіду "YearsOfExperience", опис "Description", посилання на фото "PhotoUrl", посилання на резюме "CvUrl", електронну пошту "Email" і номер телефону "Phone". Кандидат також має "SeniorityLevelId", що відповідає його рівню кваліфікації, та "CandidateOriginId", що вказує на джерело походження кандидата.

Клас "Recruiter" представляє рекрутера, відповідального за пошук кандидатів на вакансії. У нього є "Id", "FirstName", "LastName", "Email" та "PhotoUrl". Він також має зв'язок з вакансіями, якими він керує.

"Vacancy" представляє вакансію, доступну для кандидатів. У нього є унікальний "Id", назва "Title", потрібний досвід "YearsOfExperience", максимальна заробітна плата "MaxSalary", опис "Description", "AssignedTo", що вказує на рекрутера, відповідального за цю вакансію, та "StatusId" та "JobTypeId", що представляють стан вакансії та тип роботи відповідно.

Сутності "Candidate" та "Vacancy" пов'язані через "CandidateVacancy", що представляє заявку кандидата на вакансію, та "Interview", що представляє співбесіду між кандидатом та рекрутером.

"SeniorityLevel", "CandidateOrigin", "JobType" та "Status" - це допоміжні сутності, що допомагають категоризувати кандидатів та вакансії за різними критеріями.

3.4 Розробка схеми бази даних

Для відображення згаданих вище класів у схемі бази даних (рис.3.8) MS SQL використовується ряд таблиць, кожна з яких відображає відповідний клас.

Таблиця Candidate містить наступні поля.

Id INT PRIMARY KEY - первинний ключ таблиці. Він має тип INT для оптимізації розміру та швидкості обробки.

FirstName VARCHAR(50) - поле призначене для зберігання імені кандидата. Тип даних VARCHAR(50) означає, що ім'я може складатися з будь-якої комбінації символів (включаючи букви, цифри, пробіли, спецсимволи) довжиною до 50 символів.

LastName VARCHAR(50) - поле призначене для зберігання прізвища кандидата, з тими ж обмеженнями і можливостями, що і поле FirstName.

YearsOfExperience INT - поле використовується для зберігання кількості років досвіду кандидата. Тип даних INT означає, що це число.

Description TEXT - поле призначене для зберігання додаткової інформації про кандидата. Тип TEXT означає, що це може бути великий об'єм тексту.

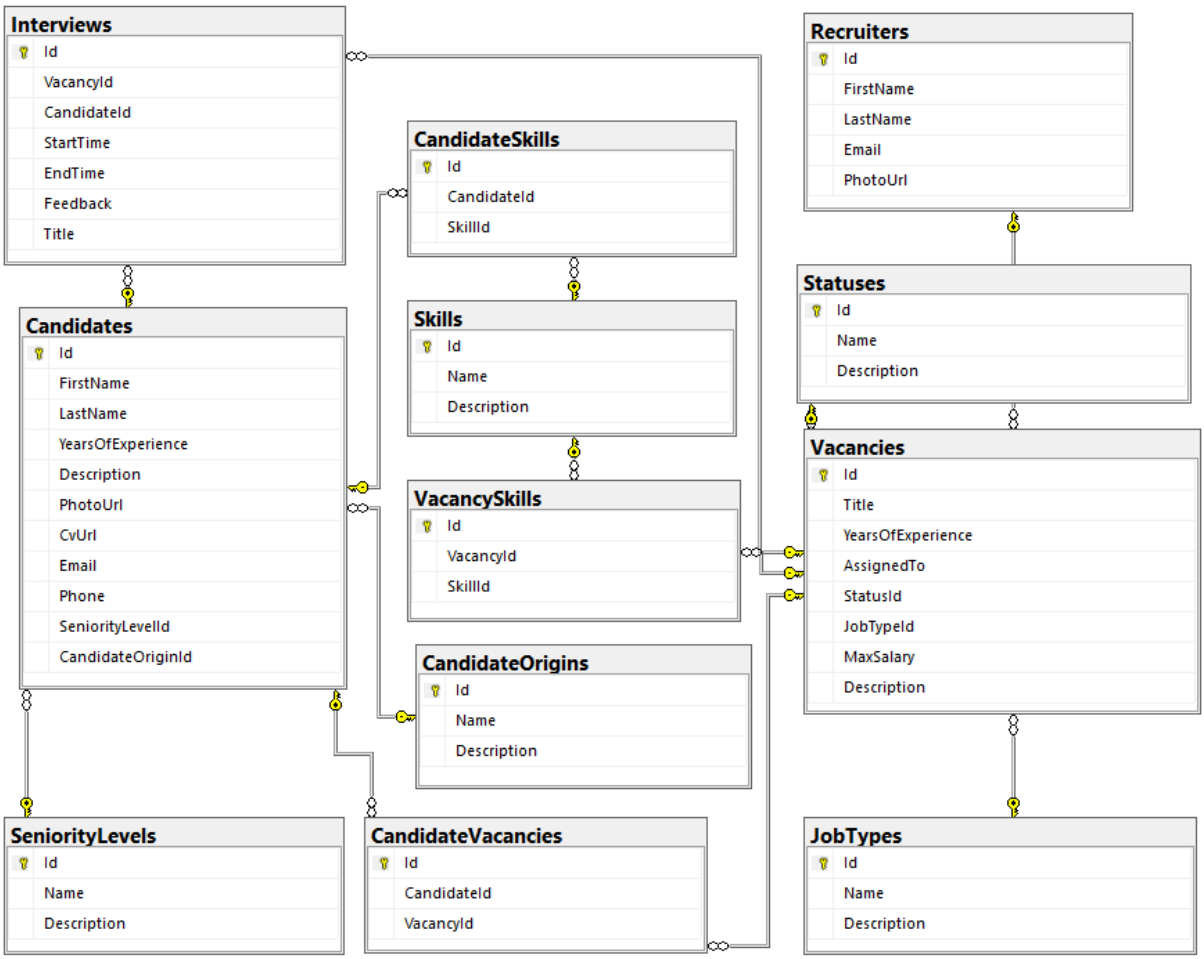


Рисунок 3.8 - Схема бази даних

PhotoUrl VARCHAR(255) - поле містить URL-адресу фотографії кандидата. Максимальна довжина URL-адреси становить 255 символів.

CvUrl VARCHAR(255) - аналогічно полю PhotoUrl, але зберігає URL-адресу резюме кандидата.

Email VARCHAR(50) - поле призначене для зберігання електронної пошти кандидата.

Phone VARCHAR(20) - поле призначене для зберігання номера телефону кандидата.

SeniorityLevelId INT FOREIGN KEY - це зовнішній ключ, який зв'язує цю таблицю з таблицею SeniorityLevel. Це означає, що значення в цьому полі відповідає рядку в таблиці SeniorityLevel, що вказує на рівень кваліфікації кандидата.

CandidateOriginId INT FOREIGN KEY - це зовнішній ключ, який зв'язує цю таблицю з таблицею CandidateOrigin. Це означає, що значення в цьому полі відповідає рядку в таблиці CandidateOrigin, що вказує на джерело, з якого кандидат був залучений.

Таблиця Recruiter містить наступні поля.

Id INT PRIMARY KEY - унікальний ідентифікатор для кожного рекрутера в базі даних. Тип даних INT означає, що це числове поле. PRIMARY KEY вказує на те, що кожне значення в цьому полі є унікальним та не може бути NULL.

FirstName VARCHAR(50) - поле призначене для зберігання імені рекрутера. Тип даних VARCHAR(50) означає, що ім'я може складатися з будь-якої комбінації символів довжиною до 50 символів.

LastName VARCHAR(50) - поле призначене для зберігання прізвища рекрутера, з тими ж обмеженнями і можливостями, що і поле FirstName.

Email VARCHAR(50) - поле призначене для зберігання електронної пошти рекрутера.

PhotoUrl VARCHAR(255) - поле містить URL-адресу фотографії рекрутера. Максимальна довжина URL-адреси становить 255 символів.

Таблиця Vacancy містить наступні поля.

Id INT PRIMARY KEY - унікальний ідентифікатор вакансії в базі даних.

Title VARCHAR(100) - поле призначене для зберігання назви вакансії.

YearsOfExperience INT - поле використовується для зберігання мінімальної кількості років досвіду, необхідної для вакансії.

MaxSalary SMALLINT - поле використовується для зберігання максимальної зарплати, яка пропонується за вакансію.

Description TEXT - поле призначене для зберігання детального опису вакансії.

AssignedTo INT FOREIGN KEY - зовнішній ключ, який вказує на відповідний рядок в таблиці Recruiter. За допомогою цього поля можна встановити, хто відповідає за конкретну вакансію.

StatusId INT FOREIGN KEY - зовнішній ключ, який зв'язує цю таблицю з таблицею Status. Це означає, що значення в цьому полі відповідає рядку в таблиці Status, що вказує на статус вакансії.

JobTypeId INT FOREIGN KEY - зовнішній ключ, який зв'язує цю таблицю з таблицею JobType. Це означає, що значення в цьому полі відповідає рядку в таблиці JobType.

Зв'язки багато-до-багатьох між таблицями Candidate та Vacancy реалізовані за допомогою двох допоміжних таблиць CandidateVacancy та Interview. Вони мають відповідні зовнішні ключі, які посилаються на первинні ключі Candidate та Vacancy.

Таблиці SeniorityLevel, CandidateOrigin, JobType, Status мають первинний ключ INT та відповідні поля для зберігання інформації.

Для усіх текстових полів використовується тип VARCHAR з відповідним обмеженням довжини для оптимізації використання простору. Всі зовнішні ключі вказують на первинні ключі відповідних таблиць, що гарантує цілісність даних.

У базі даних MS SQL кожна з таблиць має унікальний первинний ключ, який ідентифікує кожний запис у таблиці. Поля з типом INT, що використовуються як первинні ключі, забезпечують високу продуктивність для операцій пошуку та вставки.

Для текстових полів, таких як FirstName, LastName, Email, PhotoUrl та Description, використовується VARCHAR, який забезпечує гнучкість відносно довжини записів. Розмір VARCHAR встановлено в межах розумного обмеження для кожного поля, щоб оптимізувати використання простору та зберегти цілісність даних.

Зв’язки між таблицями встановлюються за допомогою зовнішніх ключів, які гарантують цілісність даних у взаємозв'язаних таблицях. Наприклад, SeniorityLevelId у таблиці Candidate посилається на запис в таблиці SeniorityLevel. Це означає, що для кожного кандидата в базі даних повинен існувати відповідний запис про рівень кваліфікації в таблиці SeniorityLevel.

Для відображення взаємин багато-до-багатьох між кандидатами та вакансіями використовуються допоміжні таблиці CandidateVacancy та Interview. Вони містять зовнішні ключі, які вказують на відповідні записи в таблицях Candidate та Vacancy, забезпечуючи трекінг, які кандидати подавались на які вакансії, а також які співбесіди були проведені.

Таблиці SeniorityLevel, CandidateOrigin, JobType, Status дозволяють зберігати нормалізовані дані, що сприяє зменшенню дублікації і полегшує обслуговування та масштабування бази даних.

Кожна з таблиць в схемі бази даних MS SQL відображає відповідний клас у .NET Core API. Це сприяє узгодженості між базою даних і бізнес-логікою, а також полегшує розуміння структури даних. [12]

3.5 Розробка бекенд частини за допомогою .NET Core

Створення проекту включає в себе декілька основних етапів.

Ініціалізація проекту. Використовуючи команду dotnet new, створюємо новий проект .NET Core Web API: dotnet new webapi -n OnboardingTrackerWebAPI.

Організація структури проекту представлена на рис.3.9. Згідно з Onion Architecture, необхідно розділити проект на наступні шари: доменний (середина цибулини), доменний інтерфейс, інфраструктура та API.

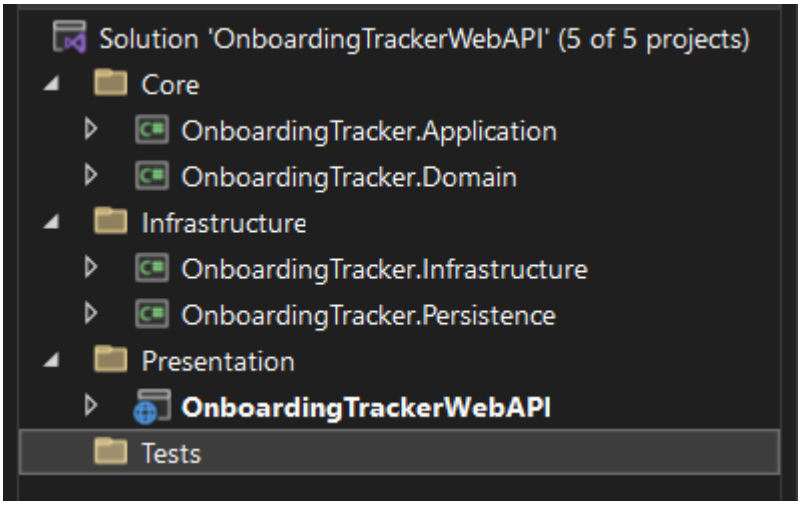


Рисунок 3.9 - Структура проекту

Встановлення потрібних пакетів. Проект потребує певного набору пакетів, зокрема: MediatR, Entity Framework, AutoMapper, Swagger, Auth0 SDK. Встановлення відбувається за допомогою команди dotnet add package.

Конфігурація Swagger. Swagger дозволяє легко створювати документацію API, яку можна переглядати через веб-інтерфейс. Конфігурація Entity Framework забезпечує просту та ефективну взаємодію з базою даних.Використання CQRS з MediatR. CQRS (Command Query Responsibility Segregation) разом з MediatR дозволяє розділити логіку читання та запису, що допомагає управляти складністю системи. Інтеграція Auth0. Auth0 забезпечує безпечну автентифікацію та авторизацію для API.

Процес кешування відіграє важливу роль у зниженні навантаження на базу даних.

Результатом розробки є .NET Core Web API, створений з використанням Onion Architecture, який використовує CQRS з MediatR, інтегрований з Swagger та Auth0, з Entity Framework як ORM, та реалізацією процесу кешування.

3.6 Розробка фронтед частини за допомогою Angular

Проект, створений на основі Angular, представляє собою односторінковий додаток (SPA), в якому вміст динамічно оновлюється без потреби перезавантаження сторінки. Для надання додатку стильного і зручного вигляду використовуєтьсяAngular Material, Tailwind CSS та Bootstrap.

Побудова проекту Angular з нуля не є тривіальною задачею і включає в себе кілька ключових етапів. На першому етапі відбувається ініціалізація проекту за допомогою Angular CLI. За допомогою команди "ng new" створюється новий проект.

На наступному етапі встановлюються потрібні для проєкту бібліотеки. Щоб надати гарний вигляд додатку та взаємодіяти з користувачем, встановлюються Angular Material, Tailwind CSS та Bootstrap за допомогою npm.

Наступна важлива частина проєкту - інтеграція Auth0. Auth0 забезпечує безпечну аутентифікацію та авторизацію для Angular додатку. Після встановлення Auth0 SDK та його налаштовування наступним етапом є реалізація необхідних компонентів, які відображають і забезпечують взаємодію з функціоналом, що був реалізований на бекенді. У класі компонента викликаються методи з сервісів, що дозволяє виконувати запити до API.

Не менш важливим є обробка помилок, які можуть виникнути в процесі взаємодії з нашим API. Для цього створюється Interceptor. В шаблоні головного компонента використовується translate pipe, яка слугує для реалізації локалізації в додатку.

Для підтримки української та англійської мов створено два .po файли. Angular використовує ці файли для перекладу тексту в додатку за допомогою translate pipe.

Наступний важливий етап - підключення Reactive Forms. Цей модуль дозволяє створювати складні форми з динамічною логікою, включаючи валідацію та керування стану на клієнтській стороні. [5]

Також використовується Angular Router для реалізації навігації в односторінковому додатку. Цей потужний інструмент дозволяє задавати маршрути та організувати перехід між різними компонентами додатку.

Одним з ключових елементів оптимізації завантаження додатку є використання lazy loading. Завдяки цьому, модулі додатку завантажуються лише при першому відвідуванні відповідної сторінки, що значно зменшує час завантаження додатку.

Бібліотеку RxJS використовується для реалізації асинхронних операцій. За допомогою Observable, Subject та операторів RxJS в проєкті можна створювати складні асинхронні потоки даних. [7]

3.7 Unit тестування

Unit-тестування є невід'ємною частиною сучасного процесу розробки

програмного забезпечення, включаючи Agile та DevOps. Цей метод тестування дозволяє розробникам досконало перевірити роботу кожного окремого модуля, або "юніту", в архітектурі програми, що в результаті гарантує її коректну роботу.

В проєкті використовується технологія XUnit Серед основних переваг XUnit, варто відзначити підтримку обробки виключень через атрибути Assert.Throws<> та використання атрибутів Theory та InlineData. Ці атрибути дозволяють налаштувати тестування методів з різними вхідними даними.

Робота з XUnit проходить у декілька ключових етапів. Перший - підготовка тестового середовища, включаючи ініціалізацію всіх необхідних змінних і об'єктів. Наступний етап - написання тест-кейсів, при цьому кожен кейс спрямований на перевірку конкретного аспекту функціональності. Зазвичай, кожен тест перевіряє один сценарій використання. Нарешті, проводиться запуск тестів і аналіз результатів. XUnit автоматизує цей процес, відстежуючи результати кожного тесту і генеруючи детальний звіт про їх успішність. [8]

При тестуванні бекенд частини, основна увага приділяється бізнес-логіці, а мок-об'єкти використовуються для імітації взаємодії з базою даних та іншими зовнішніми сервісами.

Далі наведено опис розроблених тест-кейсів для сутності "Recruiter" в контексті виконання CRUD операцій. Почнемо з "Create".

Перший тест називається "Validate_Positive". Його мета – переконатись, що валідатор CreateRecruiter правильно валідує вхідні дані. На вхід передаються коректні значення для імені, прізвища, електронної пошти та URL фотографії. Очікуваний результат тесту - результаті валідації не виявиться жодної помилки.

Наступний тест "Validate_Negative" дозволяє перевірити, чи вміє валідатор CreateRecruiter виявляти некоректні вхідні дані. В даному випадку, на початку передаються значення, які перевищують допустиму довжину, а після цього передаються порожні значення. Очікуваний результат тесту - валідація в обох випадках виявить помилки.

"Handle_Positive" - тест, який перевіряє, чи здатний обробник CreateRecruiter коректно створювати нових рекрутерів, коли передаються валідні

дані. Передаються тестові дані, очікуваний результат - новий рекрутер буде успішно створений на основі цих даних.

Тест "Handle_Negative_ThrowsValidationException" дозволяє переконатись, що обробник CreateRecruiter коректно реагує на недійсні вхідні дані. Передаються дані, що вже використовуються іншим рекрутером, очікуваний результат - помилка валідації.

Розглянемо тест-кейси для операції “Delete”.

Позитивний тест-кейс Handle_Positive - цей тест-кейс перевіряє сценарій, коли видаляється рекрутер, який не пов'язаний з будь-якими іншими сутностями в системі. В запит передається ідентифікатор рекрутера Id як 1, обробляється за допомогою виклику методу Handle і перевіряєється, що рекрутера з цим ідентифікатором вже не існує.

Негативний тест-кейс Handle_Negative_ThrowsNotFoundException - перевіряє сценарій спроби видалити рекрутера, якого не існує в системі. У запиті використовується ідентифікатор рекрутера Id як 0 і очікуєється, що при спробі видалити такого рекрутера буде виняток NotFoundException.

Негативний тест-кейс Handle_Negative_ThrowsValidationException - цей тест-кейс перевіряє сценарій, коли відбувається спроба видалити рекрутера, який є пов'язаним з іншою сутністю в системі. У цьому випадку в запит передається ідентифікатор рекрутера Id як 2 і очікується, що при спробі видалити такого рекрутера буде виняток ValidationException.

Розглянемо тест-кейси операції “GET”.

Перший з них Handle_Positive, фокусується на позитивному сценарії. Його основна мета – переконатися, що у випадку наявності рекрутера з конкретним ID, метод Handle здатен повернути дані про цього рекрутера. В рамках цього тесту використовується ID першого рекрутера з локальної бази даних, а на виході очікується отримання об'єкту, що еквівалентний оригінальному рекрутеру. Важливо, що результатом цього тесту не може бути null.

Тест-кейс Handle_Negative_ThrowsNotFoundException сконцентрований на негативному сценарії. Тут перевіряється випадок, коли рекрутера з вказаним ID в

локальній базі даних не знайдено. Для цього використовується ID, який не прив'язаний до жодного рекрутера (в цьому випадку, це 0). Очікується, що на цей запит метод Handle поверне обьект виключення типу NotFoundException. Цей тест важливий для перевірки того, що система коректно реагує на випадки, коли дані, які вона очікує отримати, не знайдені в базі даних.

За результатами проведеного тестування всі зазначені позитивні та негативні тести було пройдено успішно.

ВИСНОВКИ

1. Виконано огляд, аналіз та характеристику існуючих HR Onboarding платформ. Проведено детальний аналіз поточних систем HR Onboarding, що дозволило виявити ключові вимоги до таких систем та ідентифікувати популярні технології, що застосовуються для їх реалізації.

2. Вибрано бекенд і фронтенд технології для створення додатку - .NET Core і Angular відповідно. Обґрунтовано вибір архітектури для API, розглянуто механізми аутентифікації та авторизації, що є ключовими для безпеки HR Onboarding системи.

3. В ході проектування і розробки програмного продукту, було створено діаграму класів, схему бази даних, розроблено бекенд і фронтенд частини програмного продукту за допомогою вибраних технологій. Проведено Unit тестування, що забезпечує високий рівень якості коду.

4. Результати роботи показали, що розробка HR Onboarding системи є важливим та актуальним завданням. Обрані технології та архітектура дозволяють створити надійну, масштабовану та гнучку систему, що відповідає сучасним вимогам HR. Результати роботи можуть бути використані, як основа для подальшої розробки та удосконалення HR Onboarding систем.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Talmundo onboarding platform. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.zavvy.io/vs/talmundo-alternatives

2. ClickBoarding onboarding platform. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.clickboarding.com/resources/blog

3. Onboarding Gamification Examples For New Hires. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://elearningindustry.com/how-to-improve-new-hire-integration-and-engagement-onboarding-gamification-examples

4. Microsoft. .NET Core Guide. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://docs.microsoft.com/en-us/dotnet/core/

5. The Modern JavaScript Tutorial. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://javascript.info/

6. Understanding Authentication, Authorization. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.bu.edu/tech/about/security-resources/bestpractice/auth/

7. Angular. Angular Documentation. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://angular.io/docs

8. Become a Unit Test Master. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://leocoout.medium.com/become-a-unit-test-master-84f4fa276deb

9. Everything You Need to Know About UML Diagrams. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.smartdraw.com/uml-diagram/

10. Software architecture design patterns to know. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.redhat.com/architect/14-software-architecture-patterns

11. ASP.NET Core Best Practices. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/best-practices?view=aspnetcore-7.0

12. How to Draw a Database Model Diagram. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу: https://www.edrawsoft.com/how-to-draw-database-model-diagram.html

ДОДАТОК А
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка HR Onboarding платформи з використанням .NET Core Web API, MS SQL, Angular

Виконав студент 5 курсу
групи ПП3-51
Лаптєв Андрій Олексійович
Керівник роботи
К.т.н, доц, доцент кафедри ІПЗ Золотухіна Оксана Анатоліївна
Київ – 2023

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – оптимізація процесу HR Onboarding за рахунок використання платформи, створеної засобами .NET Core, Web API, MS SQL і Angular.

Об'єкт дослідження – процес HR Onboarding.

Предмет дослідження – програмне забезпечення процесу HR Onboarding.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Оцінка доступних систем HR Onboarding.
2. Дослідження технологій, які використовуються в HR Onboarding платформах.
3. Розробка технічного завдання для HR Onboarding платформи.
4. Розробка архітектури додатку.
5. Розробка структури бази даних.
6. Розробка та тестування клієнтської частини платформи за допомогою Angular.

3

АНАЛІЗ АНАЛОГІВ

	Talmundo	Sapling	Click Boarding	HR Onboarding
Переваги	Інтуїтивний інтерфейс, гнучкість, мультиязичність	Інтеграція з іншими HR системами, візуально привабливий інтерфейс	Швидке оформлення онбордингових процесів, мобільний додаток	Швидкодія платформи, простота використання
Ключові функції	Персоналізований онбординг, інтеграція з HR системами, звітність	Керування талантами, інтеграція з HR системами, звітність	Швидке оформлення онбордингових процесів, спідкування за прогресом, звітність	Управління списком кандидатів, рекрутерів та вакансій
Особливості реалізації	Хмарна модель, зручна для великих компаній	Хмарна модель, підходить для великих організацій	Хмарна модель, мобільний додаток	Onion архітектура, SPA додаток, інтеграція з Auth0
Недоліки	Немає мобільного додатку	Потребує велику кількість часу на впровадження	Обмеження в напаштуванні, відсутність мультиязичності	Обмежена функціональність

4

ВИМОГИ ДО ДОДАТКУ

Функціональні вимоги:

1. Авторизація та реєстрація користувачів: Система повинна надавати можливість створити обліковий запис, ввійти в систему, відновити забуті паролі.
2. Управління процесом онбордингу: Система повинна надавати інструменти для створення та відстеження різних еталів процесу онбордингу.
3. Керування користувачами: Система повинна дозволяти адміністраторам створювати, редагувати, блокувати та видаляти облікові записи користувачів.

Нефункціональні вимоги:

1. Безпека: всі дані користувачів та компанії повинні бути безпечно збережені.
2. Швидкодія: система повинна швидко реагувати на запити користувачів і ефективно обробляти великі обсяги даних.
3. Масштабованість: система повинна мати можливість розширюватися з підтримкою великого числа користувачів.
4. Надійність: система повинна гарантувати стабільну роботу та відсутність втрати даних.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

ANGULAR

Angular Material

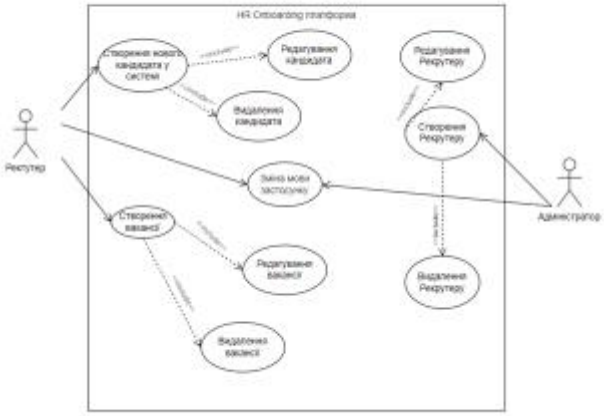
Bootstrap

tailwindcss

Swagger
Supported by SMARTBEAR

7

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

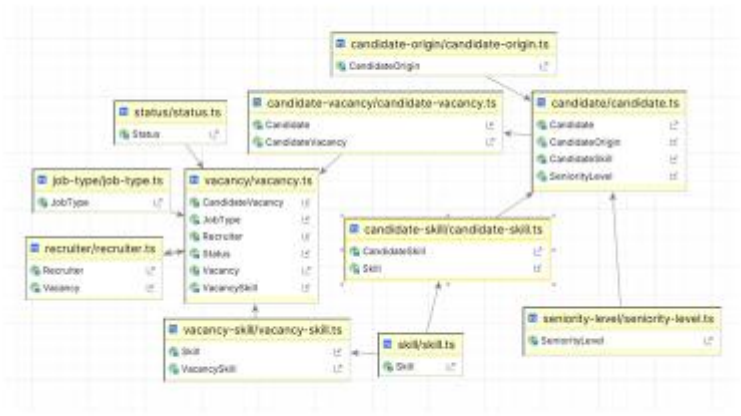


8

СХЕМА БАЗИ ДАНИХ

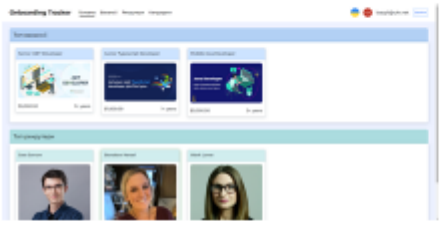
9

ДІАГРАМА КЛАСІВ

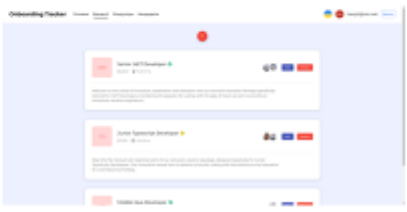


10

ЕКРАННІ ФОРМИ



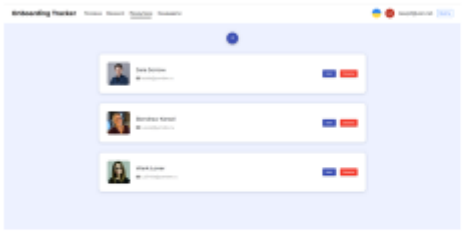
Головний екран додатку



Вакансії

11

ЕКРАННІ ФОРМИ



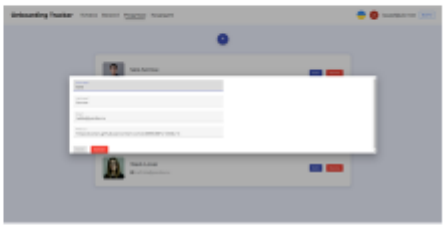
Рекрутери

Кандидати

12

ЕКРАННІ ФОРМИ

Редагування вакансії

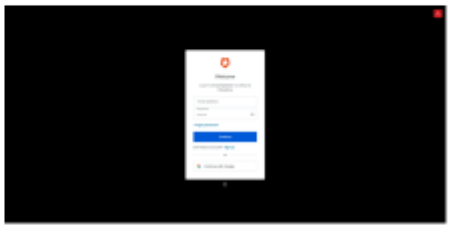


Редагування рекрутера

13

ЕКРАННІ ФОРМИ

Редагування кандидата



Логін сторінка

14

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Лаптєв А.О. Роль HR Onboarding платформ в сучасних компаніях / О.А.Золотухіна, А.О. Лаптєв // Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії: Матеріали міжнародної науково-технічної конференції. Державний університет телекомунікацій, м.Київ, 01- 03 червня 2023 р. Подано до друку.
2. Лаптєв А.О. Розгляд та порівняння популярних HR Onboarding платформ на ринку // О.А.Золотухіна, А.О. Лаптєв // Сучасні аспекти діджиталізації та інформатизації в програмній та комп'ютерній інженерії: Матеріали міжнародної науково-технічної конференції. Державний університет телекомунікацій, м.Київ, 01- 03 червня 2023 р. Подано до друку.

15

ВИСНОВКИ

1. Проведено аналіз існуючих систем для HR Onboarding. Виявлено, що багато з них мають складний інтерфейс, високу вартість. Водночас, перевагою більшості систем є здатність до інтеграції з іншими системами HR та кросплатформеність.
2. Проведено огляд та аналіз існуючих технологій, що використовуються для розробки HR Onboarding платформ. Ключовими критеріями для оцінки були надійність, продуктивність та можливість інтеграції.
3. Сформовано технічне завдання на розробку HR Onboarding платформи. Основні елементи системи включають дизайн інтерфейсу, базу даних для зберігання інформації, а також API для можливої інтеграції з іншими HR системами.
4. Розроблено архітектуру додатку.
5. Розроблено структуру бази даних використовуючи MS SQL, що дозволяє зберігати інформацію про процеси і легко інтегруватись з іншими системами.
6. Розроблено клієнтську частину платформи за допомогою Angular, що забезпечує гнучкість та адаптивність інтерфейсу.

ДОДАТОК Б

ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

Backend частина:

```
// Program.cs файл с основною конфігурацією API
public class Program
{
   public static void Main(string[] args)
   {
      var builder = WebApplication.CreateBuilder(args);
      var configuration = builder.Configuration;


builder.Services.AddControllers().AddNewtonsoftJson(options =>

options.SerializerSettings.ReferenceLoopHandling = ReferenceLoopHandling.Ignore);


builder.Services.AddDistributedMemoryCache();

      builder.Services.AddScoped<IUserContext, UserContext>();

      builder.Services.AddApiVersioning(options =>
      {

options.AssumeDefaultVersionWhenUnspecified = true;
         options.DefaultApiVersion = ApiVersion.Default;
      });
      builder.Services.AddVersionedApiExplorer();
      builder.Services.AddHealthChecks();

      builder.Services.AddMediatR(cfg => cfg.RegisterServicesFromAssemblyContaining<GetCandidateById>());

builder.Services.AddAutoMapper(typeof(GetCandidatesByFilter).Assembly);

builder.Services.AddValidatorsFromAssembly(typeof(GetCandidatesByFilter).Assembly);

builder.Services.AddDbContext(configuration);
      builder.Services.AddSwagger(configuration);

builder.Services.AddAuthentication(configuration);


builder.Services.AddTransient(typeof(IPipelineBehavior<,>), typeof(RequestValidationBehaviour<,>));

builder.Services.AddTransient(typeof(IRequestPreProcessor<>), typeof(RequestLoggerBehaviour<>));

      var allowedOrigins = configuration.GetSection("CorsAllowedOrigins").Get<IEnumerable<string>>();
      builder.Services.AddCors(options =>
      {
         options.AddDefaultPolicy(builder =>
         {

builder.WithOrigins(allowedOrigins.ToArray()).AllowAnyHeader().AllowAnyMethod();
         });
      });


builder.Services.Configure<OAuth2Options>(configuration.GetSection(OAuth2Options.SectionName));

builder.Services.Configure<CachingOptions>(configuration.GetSection(CachingOptions.SectionName));

builder.Services.Configure<DbOptions>(configuration.GetSection(DbOptions.SectionName));

      using var app = builder.Build();


app.UseMiddleware<GlobalExceptionMiddleware>();

      app.MigrateDatabase(configuration);
      app.UseSwaggerCustom(configuration);
```

```
        app.UseHttpsRedirection();
        app.UseStaticFiles();

        app.UseRouting();
        app.UseCors();

        app.UseAuthentication();
        app.UseAuthorization();

app.UseMiddleware<CurrentUserMiddleware>();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
            endpoints.MapHealthChecks("/health");
        });

        app.Run();
    }
}


// OnboardingTrackerDbContext.cs для роботи з EF Core
public class OnboardingTrackerDbContext : DbContext
{
    public OnboardingTrackerDbContext()
    {
    }

    public OnboardingTrackerDbContext(DbContextOptions<OnboardingTrackerDbContext> options)
        : base(options)
    {
    }

    public virtual DbSet<Candidate> Candidates { get; set; }

    public virtual DbSet<CandidateOrigin> CandidateOrigins { get; set; }

    public virtual DbSet<CandidateSkill> CandidateSkills { get; set; }

    public virtual DbSet<CandidateVacancy> CandidateVacancies { get; set; }

    public virtual DbSet<Interview> Interviews { get; set; }

    public virtual DbSet<JobType> JobTypes { get; set; }

    public virtual DbSet<Recruiter> Recruiters { get; set; }

    public virtual DbSet<SeniorityLevel> SeniorityLevels { get; set; }

    public virtual DbSet<Skill> Skills { get; set; }

    public virtual DbSet<Status> Statuses { get; set; }

    public virtual DbSet<User> Users { get; set; }

    public virtual DbSet<Vacancy> Vacancies { get; set; }

    public virtual DbSet<VacancySkill> VacancySkills { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {

modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());

        modelBuilder.Seed();

        base.OnModelCreating(modelBuilder);
    }
}


// Конфігурація таблиці кандидату
public class CandidateConfiguration : IEntityTypeConfiguration<Candidate>
{
    public void Configure(EntityTypeBuilder<Candidate> builder)
    {
        builder.Property(e => e.CvUrl).HasMaxLength(2083);

        builder.Property(e => e.Description).HasMaxLength(500);
```

```
        builder.Property(e => e.Email)
            .IsRequired()
            .HasMaxLength(150);

        builder.Property(e => e.FirstName)
            .IsRequired()
            .HasMaxLength(150);

        builder.Property(e => e.LastName)
            .IsRequired()
            .HasMaxLength(150);

        builder.Property(e => e.Phone).HasMaxLength(150);

        builder.Property(e => e.PhotoUrl).HasMaxLength(2083);

        builder.HasOne(d => d.CandidateOrigin)
            .WithMany(p => p.Candidates)
            .HasForeignKey(d => d.CandidateOriginId)
            .OnDelete(DeleteBehavior.Restrict)

.HasConstraintName("FK_Candidate_CandidateOrigin");

        builder.HasOne(d => d.SeniorityLevel)
            .WithMany(p => p.Candidates)
            .HasForeignKey(d => d.SeniorityLevelId)
            .OnDelete(DeleteBehavior.Restrict)

.HasConstraintName("FK_Candidate_SeniorityLevel");
    }
}


// CandidatesController.cs
public class CandidatesController : ApiBaseController
{
    public CandidatesController(IMediator mediator)
        : base(mediator)
    {
    }

    /// <summary>
    /// Get candidates with paging and filtering.
    /// </summary>
    [HttpGet]
    [ProducesResponseType(typeof(CandidatesList), StatusCodes.Status200OK)]
    public async Task<ActionResult> Get([FromQuery]GetCandidatesByFilter query)
    {
        return Ok(await Mediator.Send(query, HttpContext.RequestAborted));
    }

    /// <summary>
    /// Get candidate by Id.
    /// </summary>
    [HttpGet("{id}")]

[ProducesResponseType(typeof(CandidateModel), StatusCodes.Status200OK)]
    public async Task<ActionResult> GetById(int id)
    {
        return Ok(await Mediator.Send(new GetCandidateById { Id = id }, HttpContext.RequestAborted));
    }

    /// <summary>
    /// Create candidate.
    /// </summary>
    [HttpPost]

[ProducesResponseType(typeof(CandidateModel), StatusCodes.Status201Created)]
    public async Task<ActionResult> Create([FromBody] CreateCandidate command)
    {
        var createdCandidate = await Mediator.Send(command);

        return CreatedAtAction(nameof(GetById), new { id = createdCandidate.Id }, createdCandidate);
    }

    /// <summary>
    /// Update candidate.
    /// </summary>
    [HttpPut]

[ProducesResponseType(typeof(CandidateModel), StatusCodes.Status202Accepted)]
    public async Task<ActionResult> Update([FromBody] UpdateCandidate command)
    {
        var updatedCandidate = await Mediator.Send(command);
```

```
        return AcceptedAtAction(nameof(GetById), new { id = updatedCandidate.Id }, updatedCandidate);
    }

    /// <summary>
    /// Delete candidate.
    /// </summary>
    [HttpDelete("{id}")]

[ProducesResponseType(StatusCodes.Status200OK)]
    public async Task<ActionResult> Delete(int id)
    {
        await Mediator.Send(new DeleteCandidate { Id = id });

        return NoContent();
    }
}


// GetCandidateById.cs
public class GetCandidateById : IRequest<CandidateModel>
{
    public int Id { get; set; }

    public class Handler : IRequestHandler<GetCandidateById, CandidateModel>
    {
        private readonly OnboardingTrackerDbContext dbContext;
        private readonly IMapper mapper;

        public Handler(OnboardingTrackerDbContext dbContext, IMapper mapper)
        {
            this.dbContext = dbContext;
            this.mapper = mapper;
        }

        public async Task<CandidateModel> Handle(GetCandidateById request, CancellationToken cancellationToken)
        {
            var candidate = await dbContext.Candidates
                .Include(x => x.SeniorityLevel)
                .Include(x => x.CandidateOrigin)
                .Include(x => x.CandidateSkills)
                .ThenInclude(x => x.Skill)
                .AsNoTracking()
                .FirstOrDefaultAsync(x => x.Id == request.Id, cancellationToken);

            if (candidate == null)
            {
                throw new NotFoundException(nameof(Candidate), request.Id);
            }

            return mapper.Map<CandidateModel>(candidate);
        }
    }
}


// CreateCandidate.cs
public class CreateCandidate : IRequest<CandidateModel>
{
    public string FirstName { get; set; }

    public string LastName { get; set; }

    public int YearsOfExperience { get; set; }

    public string Description { get; set; }

    public string PhotoUrl { get; set; }

    public string CvUrl { get; set; }

    public string Email { get; set; }

    public string Phone { get; set; }

    public int SeniorityLevelId { get; set; }

    public int CandidateOriginId { get; set; }

    public IEnumerable<int> Skills { get; set; }

    public class Validator : AbstractValidator<CreateCandidate>
    {
        public Validator()
        {
            RuleFor(x => x.FirstName).NotEmpty().MaximumLength(150);
```

```
        RuleFor(x => x.LastName).NotEmpty().MaximumLength(150);
        RuleFor(x => x.YearsOfExperience).NotEmpty().GreaterThan(0).LessThan(70);
        RuleFor(x => x.Description).MaximumLength(500);
        RuleFor(x => x.PhotoUrl).MaximumLength(2083);
        RuleFor(x => x.CvUrl).MaximumLength(2083);
        RuleFor(x => x.Email).NotEmpty().MaximumLength(150).EmailAddress();
        RuleFor(x => x.Phone).PhoneNumber();
        RuleFor(x => x.SeniorityLevelId).NotEmpty().GreaterThan(0);
        RuleFor(x => x.CandidateOriginId).NotEmpty().GreaterThan(0);
    }
}

public class Handler : IRequestHandler<CreateCandidate, CandidateModel>
{
    private readonly OnboardingTrackerDbContext dbContext;
    private readonly IMapper mapper;

    public Handler(OnboardingTrackerDbContext dbContext, IMapper mapper)
    {
        this.dbContext = dbContext;
        this.mapper = mapper;
    }

    public async Task<CandidateModel> Handle(CreateCandidate request, CancellationToken cancellationToken)
    {
        if (await dbContext.Candidates.AnyAsync(x => x.Email == request.Email, cancellationToken))
        {
            throw new ValidationException($"Candidate with email '{request.Email}' already exists");
        }

        if (!await dbContext.SeniorityLevels.AnyAsync(x => x.Id == request.SeniorityLevelId, cancellationToken))
        {
            throw new NotFoundException(nameof(SeniorityLevel), request.SeniorityLevelId);
        }

        if (!await dbContext.CandidateOrigins.AnyAsync(x => x.Id == request.CandidateOriginId, cancellationToken))
        {
            throw new NotFoundException(nameof(CandidateOrigin), request.CandidateOriginId);
        }

        var candidate = new Candidate
        {
            FirstName = request.FirstName,
            LastName = request.LastName,
            YearsOfExperience = request.YearsOfExperience,
            Description = request.Description,
            PhotoUrl = request.PhotoUrl,
            CvUrl = request.CvUrl,
            Email = request.Email,
            Phone = request.Phone,
            SeniorityLevelId = request.SeniorityLevelId,
            CandidateOriginId = request.CandidateOriginId,
        };

        await using (var transaction = await dbContext.Database.BeginTransactionAsync(cancellationToken))
        {
            dbContext.Candidates.Add(candidate);
            await dbContext.SaveChangesAsync(cancellationToken);

            // update intermediate table
            if (request.Skills != null && request.Skills.Any())
            {
                var candidateSkillsToAdd = request.Skills
                    .Select(skillId => new CandidateSkill { CandidateId = candidate.Id, SkillId = skillId });
```

```
dbContext.CandidateSkills.AddRange(candidateSkillsToAdd);

                await dbContext.SaveChangesAsync(cancellationToken);
            }

            await transaction.CommitAsync(cancellationToken);
        }

        var createdCandidate = await dbContext.Candidates
            .Include(x => x.CandidateSkills).ThenInclude(x => x.Skill)
            .Include(x => x.SeniorityLevel)
            .Include(x => x.CandidateOrigin)
            .FirstOrDefaultAsync(x => x.Id == candidate.Id, cancellationToken);

        return mapper.Map<CandidateModel>(createdCandidate);
      }
   }
}


// DeleteCandidate.cs
public class DeleteCandidate : IRequest
{
   public int Id { get; set; }

   public class Handler : IRequestHandler<DeleteCandidate>
   {
      private readonly OnboardingTrackerDbContext dbContext;

      public Handler(OnboardingTrackerDbContext dbContext)
      {
         this.dbContext = dbContext;
      }

      public async Task Handle(DeleteCandidate request, CancellationToken cancellationToken)
      {
         var candidate = await dbContext.Candidates.FindAsync(request.Id);

         if (candidate == null)
         {
            throw new NotFoundException(nameof(Candidate), request.Id);
         }

         if (await dbContext.Interviews.AnyAsync(x => x.CandidateId == request.Id, cancellationToken))
         {
            throw new ValidationException("There is existing interview associated with this candidate");
         }

         dbContext.Candidates.Remove(candidate);
         await dbContext.SaveChangesAsync(cancellationToken);
      }
   }
}


// UpdateCandidate.cs
public class UpdateCandidate : IRequest<CandidateModel>
{
   public int Id { get; set; }

   public string FirstName { get; set; }

   public string LastName { get; set; }

   public int YearsOfExperience { get; set; }

   public string Description { get; set; }

   public string PhotoUrl { get; set; }

   public string Email { get; set; }

   public string Phone { get; set; }

   public int SeniorityLevelId { get; set; }

   public int CandidateOriginId { get; set; }

   public IEnumerable<int> Skills { get; set; }

   public class Validator : AbstractValidator<UpdateCandidate>
   {
      public Validator()
```

```
    {
        RuleFor(x => x.FirstName).NotEmpty().MaximumLength(150);
        RuleFor(x => x.LastName).NotEmpty().MaximumLength(150);
        RuleFor(x => x.YearsOfExperience).NotEmpty().GreaterThan(0).LessThan(70);
        RuleFor(x => x.Description).MaximumLength(500);
        RuleFor(x => x.PhotoUrl).MaximumLength(2083);
        RuleFor(x => x.Email).NotEmpty().MaximumLength(150).EmailAddress();
        RuleFor(x => x.Phone).PhoneNumber();
        RuleFor(x => x.SeniorityLevelId).NotEmpty().GreaterThan(0);
        RuleFor(x => x.CandidateOriginId).NotEmpty().GreaterThan(0);
    }
  }

  public class Handler : IRequestHandler<UpdateCandidate, CandidateModel>
  {
    private readonly OnboardingTrackerDbContext dbContext;
    private readonly IMapper mapper;

    public Handler(OnboardingTrackerDbContext dbContext, IMapper mapper)
    {
      this.dbContext = dbContext;
      this.mapper = mapper;
    }

    public async Task<CandidateModel> Handle(UpdateCandidate request, CancellationToken cancellationToken)
    {
      var candidate = await dbContext.Candidates
        .Include(x => x.CandidateSkills)
        .FirstOrDefaultAsync(x => x.Id == request.Id, cancellationToken);

      if (candidate == null)
      {
        throw new NotFoundException(nameof(Candidate), request.Id);
      }

      if (await dbContext.Candidates.AnyAsync(x => x.Email == request.Email && x.Id != request.Id, cancellationToken))
      {
        throw new ValidationException($"Candidate with email '{request.Email}' already exists");
      }

      if (!await dbContext.SeniorityLevels.AnyAsync(x => x.Id == request.SeniorityLevelId, cancellationToken))
      {
        throw new NotFoundException(nameof(SeniorityLevel), request.SeniorityLevelId);
      }

      if (!await dbContext.CandidateOrigins.AnyAsync(x => x.Id == request.CandidateOriginId, cancellationToken))
      {
        throw new NotFoundException(nameof(CandidateOrigin), request.CandidateOriginId);
      }

      candidate.FirstName = request.FirstName;
      candidate.LastName = request.LastName;
      candidate.YearsOfExperience = request.YearsOfExperience;
      candidate.Description = request.Description;
      candidate.PhotoUrl = request.PhotoUrl;
      candidate.Email = request.Email;
      candidate.Phone = request.Phone;
      candidate.SeniorityLevelId = request.SeniorityLevelId;
      candidate.CandidateOriginId = request.CandidateOriginId;

      // update intermediate table
      if (request.Skills != null && request.Skills.Any())
      {
        var candidateSkills = request.Skills
          .Select(skillId => new CandidateSkill { CandidateId = candidate.Id, SkillId = skillId });
```

```
dbContext.RemoveRange(dbContext.CandidateSkil
ls.Where(x => x.CandidateId == candidate.Id));

dbContext.CandidateSkills.AddRange(candidateSki
lls);
        }

        await
dbContext.SaveChangesAsync(cancellationToken);

        var updatedCandidate = await
dbContext.Candidates
            .Include(x =>
x.CandidateSkills).ThenInclude(x => x.Skill)
            .Include(x => x.SeniorityLevel)
            .Include(x => x.CandidateOrigin)
            .FirstOrDefaultAsync(x => x.Id ==
candidate.Id, cancellationToken);

        return
mapper.Map<CandidateModel>(updatedCandidate)
;
      }
   }
}
```

Frontend частина:

```
// home-page.component.ts
@Component({
 selector: 'app-home',
 templateUrl: './home-page.component.html',
 styleUrls: ['./home-page.component.scss'],
 changeDetection:
ChangeDetectionStrategy.OnPush
})
export class HomePageComponent implements
OnInit {
 public recruiters$ = new
BehaviorSubject<Array<Recruiter>>(new
Array<Recruiter>());
 public vacancies$ = new
BehaviorSubject<Array<Vacancy>>(new
Array<Vacancy>());
 private destroy$ = new Subject();

 constructor(
  private vacancyService: VacancyService,
  private recruiterService: RecruiterService) {
 }

 ngOnInit(): void {
  this.recruiterService
   .getRecruiters()
   .pipe(
    takeUntil(this.destroy$),
    map(recruiterResult =>
recruiterResult.recruiters),
    tap(recruiters => {
     this.recruiters$.next(recruiters);
    }))
   .subscribe();

  this.vacancyService
   .getVacancies()
   .pipe(
    takeUntil(this.destroy$),
    map(vacancyResult =>
vacancyResult.vacancies),
    tap(vacancies => {
     this.vacancies$.next(vacancies);
    }))
   .subscribe();
 }
}
```

```
<!--home-page.component.html-->
<div class="top">
 <div class="top__section">
  <div class="top__section-head top__section-
head--blue">
   <h2 class="top__section-title">{{
'TOP__VACANCIES' | translate }}</h2>
  </div>

  <div *ngIf="vacancies$ | async as vacancies"
class="top__section-body">
   <app-vacancy-card
    *ngFor="let vacancy of vacancies"
    [vacancy]="vacancy"
   ></app-vacancy-card>
  </div>
 </div>

 <div class="top__section">
  <div class="top__section-head top__section-
head--green">
   <h2 class="top__section-title">{{
'TOP__RECRUITERS' | translate }}</h2>
  </div>
```

```
    <div *ngIf="recruiters$ | async as recruiters" class="top__section-body">
      <app-recruiter-card
        *ngFor="let recruiter of recruiters"

[appHighlightRecruiter]="recruiter.vacancies!.length > 1"
        [recruiter]="recruiter"
      ></app-recruiter-card>
    </div>
  </div>
</div>


// candidates-page.component.ts
@Component({
  selector: 'app-candidates',
  templateUrl: './candidates-page.component.html',
  styleUrls: ['./candidates-page.component.scss']
})
export class CandidatesPageComponent implements OnInit, OnDestroy {
  public candidates$ = new BehaviorSubject<Array<Candidate>>(new Array<Candidate>());
  public candidateOrigins$: Observable<Array<CandidateOrigin>>;
  public seniorityLevels$: Observable<Array<SeniorityLevel>>;
  public skills$: Observable<Array<Skill>>;
  private destroy$ = new Subject<void>();

  constructor(
    private candidateService: CandidateService,
    private skillService: SkillService,
    private seniorityLevelService: SeniorityLevelService,
    private candidateOriginService: CandidateOriginService,
    private dialog: MatDialog
  ) {
  }

  ngOnInit(): void {
    this.candidateService
      .getCandidates()
      .pipe(
        takeUntil(this.destroy$),
        map(candidateResult => candidateResult.candidates),
        tap(candidates => {
          this.candidates$.next(candidates);
        }))
      .subscribe();

    this.seniorityLevels$ = this.seniorityLevelService.getSeniorityLevels()
      .pipe(map(seniorityLevelResult => seniorityLevelResult.seniorityLevels));

    this.candidateOrigins$ = this.candidateOriginService.getCandidateOrigins()
      .pipe(map(candidateOriginResult => candidateOriginResult.candidateOrigins));

    this.skills$ = this.skillService.getSkills()
      .pipe(
        map(skillResult => skillResult.skills),
        shareReplay(1)
      );
  }

  ngOnDestroy(): void {
    this.destroy$.next();
    this.destroy$.complete();
  }

  public openUpdateDialog(candidate: Candidate): void {
    const updateDialog = this.dialog.open(CandidateDialogComponent, {
      width: '70%',
      maxHeight: '90vh',
      data: {
        candidate,
        seniorityLevels$: this.seniorityLevels$,
        candidateOrigins$: this.candidateOrigins$,
        skills$: this.skills$,
      }
    });

    updateDialog.afterClosed()
      .pipe(
        tap(updateForm => {
          if (updateForm) {
            this.onUpdateFormSubmit(updateForm, candidate);
          }
        })
      )
      .subscribe();
  }

  public openCreateDialog(): void {
```

```
  const createDialog = this.dialog.open(CandidateDialogComponent, {
    width: '70%',
    maxHeight: '90vh',
    data: {
     candidate: null,
     seniorityLevels$: this.seniorityLevels$,
     candidateOrigins$: this.candidateOrigins$,
     skills$: this.skills$,
    }
  });

  createDialog.afterClosed()
    .pipe(
     tap(createForm => {
      if (createForm) {
       this.onCreateFormSubmit(createForm);
      }
     })
    )
    .subscribe();
 }

 private onUpdateFormSubmit(updateForm: FormGroup, candidate: Candidate): void {
  const mappedUpdatedCandidate = mapFormToCandidateUpdate(candidate, updateForm);


this.candidateService.updateCandidate(mappedUpdatedCandidate)
    .pipe(
     takeUntil(this.destroy$),
     tap(updatedCandidate => {
      const updatedCandidates = this.candidates$.getValue();

updatedCandidates[updatedCandidates.indexOf(candidate)] = updatedCandidate;
      this.candidates$.next(updatedCandidates);
     }),
     map(updatedCandidate => [updatedCandidate]))
    .subscribe();
 }

 private onCreateFormSubmit(createForm: FormGroup): void {
  const mappedCreatedCandidate = mapFormToCandidateCreate(createForm);


this.candidateService.createCandidate(mappedCreatedCandidate)
    .pipe(
     takeUntil(this.destroy$),
     tap(createdCandidate => {

this.candidates$.next([...this.candidates$.getValue(), createdCandidate]);
     }),
     map(createdCandidate => [createdCandidate]))
    .subscribe();
 }
}



<!--candidates-page.component.html-->
<div class="container">
 <button
  (click)="openCreateDialog()"
  class="create-btn" color="accent"
  mat-fab>
  <mat-icon>add</mat-icon>
 </button>

 <app-candidate-row
  (onEditClick)="openUpdateDialog($event)"
  *ngFor="let candidate of candidates$ | async"
  [candidate]="candidate"
 ></app-candidate-row>
</div>



// candidate-dialog.component.ts
@Component({
 selector: 'app-candidate-dialog',
 templateUrl: './candidate-dialog.component.html',
 styleUrls: ['./candidate-dialog.component.scss']
})
export class CandidateDialogComponent implements OnInit {
 public skillsSearchControl = new FormControl();
 public filteredSkills$: Observable<Array<Skill>>;
 public form: FormGroup<any>;
 private destroy$ = new Subject();

 constructor(
  private candidateFormService: CandidateFormService,
  @Inject(MAT_DIALOG_DATA) public data: CandidateDialogData) {
 }
```

```
  ngOnInit(): void {
    if (this.data.candidate) {
      this.form = this.candidateFormService.createCandidateFormWithData(this.data.candidate);
    } else {
      this.form = this.candidateFormService.createDefaultCandidateForm();
    }

    this.filteredSkills$ = this.data.skills$
      .pipe(map(skills => skills.filter(skill =>
        (this.form.get('skills') as FormArray).controls.every(skillControl => skillControl.value.name !== skill.name))));

    this.filterSkillsOnSearchValueChanges();
  }

  public addSkill(skill: Skill): void {
    const skillControl = new FormControl(skill);
    (this.form.get('skills') as FormArray).push(skillControl);
    this.skillsSearchControl.reset();
  }

  public removeSkill(index: number): void {
    const skillsFormArray = this.form.get('skills') as FormArray;
    skillsFormArray.removeAt(index);
  }

  private filterSkillsOnSearchValueChanges(): void {
    this.skillsSearchControl.valueChanges
      .pipe(
        takeUntil(this.destroy$),
        tap(currentInput => {
          this.filteredSkills$ = this.data.skills$
            .pipe(
              map(skills => skills.filter(skill => skill.name.toLowerCase().startsWith(currentInput?.toString().toLowerCase()) &&
                (this.form.get('skills') as FormArray).controls.every(x => x.value.name !== skill.name)))
            );
        })
      ).subscribe();
  }

  public getSkills(): FormArray {
    return (this.form.get('skills') as FormArray);
  }
}
```

```
<!--candidate-dialog.component.html-->
<form [formGroup]="form" class="edit-form">
  <mat-form-field appearance="fill" class="edit-form__field">
    <mat-label>First name</mat-label>
    <input formControlName="firstName" matInput>

    <mat-error *ngIf="form.get('firstName')?.hasError('required')">
      First name is required
    </mat-error>
    <mat-error *ngIf="form.get('firstName')?.hasError('maxlength')">
      First name length is too long
    </mat-error>
  </mat-form-field>

  <mat-form-field appearance="fill" class="edit-form__field">
    <mat-label>Last name</mat-label>
    <input formControlName="lastName" matInput>

    <mat-error *ngIf="form.get('lastName')?.hasError('required')">
      Last name is required
    </mat-error>
    <mat-error *ngIf="form.get('lastName')?.hasError('maxlength')">
      Last name length is too long
    </mat-error>
  </mat-form-field>

  <mat-form-field appearance="fill" class="edit-form__field">
    <mat-label>Email</mat-label>
    <input formControlName="email" matInput>

    <mat-error *ngIf="form.get('email')?.hasError('required')">
      Email is required
    </mat-error>
```

```
    <mat-error
*ngIf="form.get('email')?.hasError('maxlength')">
      Email length is too long
    </mat-error>
    <mat-error
*ngIf="form.get('email')?.hasError('email')">
      Enter a valid email
    </mat-error>
  </mat-form-field>

  <mat-form-field appearance="fill" class="edit-
form__field edit-form__field--small">
    <mat-label>Years of experience</mat-label>
    <input class="edit-form__number-input"
formControlName="yearsOfExperience" matInput
type="number">

    <mat-error
*ngIf="form.get('yearsOfExperience')?.hasError('re
quired')">
      Years of experience is required
    </mat-error>
    <mat-error
*ngIf="form.get('yearsOfExperience')?.hasError('m
in')">
      Years of experience value must be higher
    </mat-error>
    <mat-error
*ngIf="form.get('yearsOfExperience')?.hasError('m
ax')">
      Years of experience value must be lower
    </mat-error>
  </mat-form-field>

  <mat-form-field appearance="fill" class="edit-
form__field edit-form__field--small">
    <mat-label>Phone number</mat-label>
    <input class="edit-form__number-input"
formControlName="phone" matInput type="tel">
  </mat-form-field>

  <mat-form-field appearance="fill" class="edit-
form__field">
    <mat-label>Photo url</mat-label>
    <input formControlName="photoUrl" matInput>
  </mat-form-field>


  <mat-form-field appearance="fill" class="edit-
form__field edit-form__field--small">
    <mat-label>Seniority level</mat-label>
    <mat-select
formControlName="seniorityLevel">
      <mat-option *ngFor="let seniorityLevel of
data.seniorityLevels$ | async"
[value]="seniorityLevel.id">
        {{ seniorityLevel.name }}
      </mat-option>
    </mat-select>
  </mat-form-field>

  <mat-form-field appearance="fill" class="edit-
form__field edit-form__field--small">
    <mat-label>Candidate origin</mat-label>
    <mat-select
formControlName="candidateOrigin">
      <mat-option *ngFor="let candidateOrigin of
data.candidateOrigins$ | async"
[value]="candidateOrigin.id">
        {{ candidateOrigin.name }}
      </mat-option>
    </mat-select>
  </mat-form-field>

  <mat-form-field appearance="fill" class="edit-
form__field edit-form__field--big">
    <mat-label>Description</mat-label>
    <textarea cdkTextareaAutosize
formControlName="description"
matInput></textarea>

    <mat-error
*ngIf="form.get('description')?.hasError('maxlength
')">
      Description length is too long
    </mat-error>
  </mat-form-field>

  <mat-form-field appearance="fill" class="edit-
form__field">
    <mat-label>Skills</mat-label>
    <input [formControl]="skillsSearchControl"
        [matAutocomplete]="auto"
        aria-label="Skills"
        matInput
        placeholder="Pick skill to add"
        type="text">
    <mat-autocomplete #auto="matAutocomplete"
autoActiveFirstOption>
      <mat-option
        (click)="addSkill(skill)"
        *ngFor="let skill of filteredSkills$ | async"
        [value]="">
```

```
      {{skill.name}}
     </mat-option>
   </mat-autocomplete>
   <mat-error
*ngIf="form.get('skills')?.hasError('maxControlsCo
untValidator')">
     You can't add more than 5 skills
   </mat-error>
  </mat-form-field>

  <div *ngFor="let addedSkill of
getSkills().controls; index as i" class="added-
skills">
   <mat-form-field appearance="fill" class="edit-
form__field edit-form__field--small">
    <div class="d-flex">
     <input [readonly]="true"
[value]="addedSkill.value.name" matInput
type="text">
     <button (click)="removeSkill(i)" mat-icon-
button>
      <mat-icon>delete</mat-icon>
     </button>
    </div>
   </mat-form-field>
  </div>

  <div class="control-buttons control-buttons--
margin-top">
   <button [disabled]="form.invalid"
       [mat-dialog-close]="form" class="control-
button"
       color="primary"
       mat-raised-button
       type="submit">
    Save
   </button>
   <button class="control-button" color="warn"
       mat-dialog-close=""
       mat-raised-button
       type="button">
    Cancel
   </button>
  </div>
</form>


// candidate.service.ts
@Injectable({
  providedIn: 'root'
})
export class CandidateService {
  private apiUrl =
'https://localhost:7079/api/candidates';

  constructor(private http: HttpClient) {
  }

  getCandidates(): Observable<CandidateResult> {
   const params = new HttpParams()
    .set('pageNumber', '1')
    .set('pageSize', '100');

   return this.http.get<CandidateResult>(this.apiUrl,
{params});
  }

  updateCandidate(candidate: CandidateUpdate):
Observable<Candidate> {
   return this.http.put<Candidate>(this.apiUrl,
candidate);
  }

  createCandidate(candidate: CandidateCreate):
Observable<Candidate> {
   return this.http.post<Candidate>(this.apiUrl,
candidate);
  }

  deleteCandidate(id: number): Observable<void> {
   return
this.http.delete<void>(`${this.apiUrl}/${id}`);
  }
}
```