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

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

РОЗПОДІЛЕНА ВЕБСИСТЕМА ДЛЯ ОБЛІКУ ПРОФЕСІЙНИХ НАВИЧОК

Виконав студент 4-го курсу
Євген ЄРІС

(підпис)

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

(підпис)

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

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування

«_05_» __червня_____ 2023 р.,

протокол № _18__
Завідувач кафедри
Микола НІКІТЧЕНКО

(підпис)

Київ − 2023

РЕФЕРАТ

Обсяг роботи 69 сторінок, 49 ілюстрацій, 3 таблиці, 20 джерел посилань.

ПРОФЕСІЙНІЙНИЙ РОЗВИТОК, МІКРОСЕРВІСИ, ВЕБСИСТЕМА, ПАТТЕРН ПРОЄКТУВАННЯ, .NET, ANGULAR.

Об’єктом розробки є професійні навички та їх набуття. Предметом роботи є система для обліку професійних навичок.

Метою даної кваліфікаційної роботи є розроблення застосунку для збереження, систематизації й оцінки навичок користувачів із застосуванням розподіленої архітектури.

Методи розроблення: аналіз предметної області, проєктування архітектури системи, мікросервісна архітектура, DDD, CQRS, ООП із застосуванням паттернів GoF. Інструменти розроблення: інтегровані середовища розробки Visual Studio 2022 і Visual Studio Code, СКБД SQL Server Management Studio, інструмент MongoDBCompass. Бекенд функціонує на платформі .NET, графічний інтерфейс створений за допомогою Angular.

Результат роботи: виконано огляд наявних на ринку систем, що пропонують рішення в сфері управління талантами, розвитку навичок, пошуку роботи й створення резюме. Проведено аналіз переваг і недоліків мікросервісної архітектури та популярних підходів до її реалізації. Створено систему для обліку професійних навичок, яка складається із трьох сервісів REST API, компонента Gateway і клієнта Angular.

Розроблена система може застосовуватися для супроводження особистого навчання, систематизації уявлень про власні знання та створення резюме. Крім того, реалізовані API можуть використовуватися іншими клієнтами в якості серверної частини.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

BLL – Business Logic Layer, компонент програми, який інкапсулює бізнес-логіку застосунку та керує його поведінкою.

CQRS – Command and Query Responsibility Segregation, паттерн проєктування.

CV – Curriculum Vitae, резюме кандидата.

DAL – Data Access Layer, компонент програми, відповідальний за зв’язок зі сховищем даних.

DDD – Domain Driven Design, підхід до розробки програмних систем

GoF – Gang of Four, група авторів книги Design Patterns: Elements of Reusable Object-Oriented Software.

GUI – Graphic User Interface, графічний інтерфейс користувача.

SPA – Single Page Application, застосунок, що функціонує у вигляді єдиної динамічно оновлюваної сторінки.

PL – Presentation Layer, компонент програми, що є її графічним або програмним інтерфейсом.

REST – Representational State Transfer, архітектурний стиль проєктування вебсервісів.

SEO – Search Engine Optimization, практика оптимізації вмісту вебсайту для покращення його видимості в пошукових системах.

SoC – Separation of Concernes, принцип поділу інтересів у програмуванні.

SOLID – Single responsibility, Open-closed, Liskov substitution, Interface segregation, and Dependency inversion principle, п’ять принципів об’єктно-орієнтованого програмування.

БД – База Даних, постійне сховище структурованих даних.

ЕОМ – електронно-обчислювальна машина.

СКБД – система керування базами даних.

ВСТУП

Оцінка сучасного стану об’єкта розробки. Вимоги до підтримуваності, доступності, безпеки й масштабованості застосунків призвели до розвитку численних архітектурних підходів до їх створення. Розробка сучасної розширюваної системи спирається на дотриманні принципів, переважно пов’язаних із ізоляцією відповідальності її компонентів і декомпозицією вирішуваної задачі. Серед таких підходів – використання розподілених архітектур, таких як мікросервісна. При проєктуванні комерційних програмних систем сьогодення активно застосовують підходи, близькі до DDD, які забезпечують реалізацію критичних аспектів системи в тісній взаємодії з фахівцями предметної області.

Актуальність роботи та підстави для її виконання. Сьогодні онлайн-освіта є як ніколи розвинутою та популярною. Вона стала доступним способом здобуття нових компетентностей для представників будь-яких сфер діяльності. Вивчаючи нові дисципліни, важливо безперервно вести облік набутих знань. Це дозволяє систематизувати процес навчання, полегшити його планування й не дає знанням забутися.

Для онлайн-сервісів пошуку роботи характерна висока конкуренція серед кандидатів, зокрема, в IT-сфері. Статистика платформи Djinni [1] свідчить, що станом на квітень 2023 кількість претендентів перевищує загальне число вакансій майже в 6 разів. Аби мати шанси отримати перевагу над іншими кандидатами, важливо мати повне висвітлення своїх навичок і досвіду в резюме. Доцільним при цьому є використання інтернет-ресурсів, які дозволяють систематизувати й презентувати уміння користувача.

Мета й завдання роботи. Метою даної кваліфікаційної роботи є створення застосунку для збереження, систематизації й оцінки навичок користувачів із застосуванням розподіленої архітектури.

Для досягнення поставленої мети мають бути виконані такі завдання:

- дослідження предметної області й конкурентів на ринку;
- аналіз принципів, переваг і недоліків розподіленої архітектури;
- аналіз підходів до розробки індивідуальних мікросервісів;
- проєктування системи й реалізація її функцій;
- створення баз даних й імплементація логіки компонентів системи;
- дизайн GUI та розробка SPA-клієнта.

Об’єкт, методи й засоби розроблення. Об’єктом розробки застосунку є професійні навички й процеси їх здобуття, систематизації, оцінки й презентації. Предметом роботи є система для обліку професійних навичок, що складається з трьох REST API, кожен із яких має власну БД, а також SPA-клієнта.

Система була спроєктована із застосуванням мікросервісної архітектури. Для створення сервісів SkillApp і Identity використана архітектура «портів і адаптерів». Реалізація логіки застосунку виконана, зокрема, застосовуючи шаблони CQRS, «Робоча одиниця», «Специфікація», «Репозиторій», «Ланцюжок обов’язків», «Медіатор» і «Шаблонний метод» [2].

Серверна частина застосунку розроблена на базі .NET 6, використовуючи мову C#. Сервісні API створені за допомогою бібліотеки ASP.NET, їх OpenAPI специфікація згенерована інструментом Swagger. Реляційні бази даних спроєктовані в рамках підходу Code First фреймворку Entity Framework. Для сервісу CVGenerator, що представляє технологію Minimal API, використана NoSQL база даних MongoDB. Розроблений вебклієнт є застосунком типу SPA, створеним за допомогою Angular, спираючись на мову програмування TypeScript і бібліотеку RxJS. Робота інтерактивних елементів GUI забезпечена можливостями бібліотеки

Angular Material. Уніфікований інтерфейс серверних API сконфігурований з використанням Ocelot. Міжсервісна взаємодія забезпечена gRPC.

Можливі сфери застосування. Розроблений вебінтерфейс застосунку може бути застосований здобувачами знань для ведення обліку власних навичок та рівня володіння ними. Можливість перегляду профілів інших користувачів і оцінки їх умінь підкріплює використання застосунку групою людей, що вивчають спільні дисципліни. Завдяки опціям експорту даних профілю у вигляді резюме система застосовна у сфері пошуку роботи. Можливість знаходження людей за їх компетентностями може бути використана для пошуку співробітників або однодумців.

Розроблені API, що пропонують інтерфейс для взаємодії через HTTPS, можуть бути викликані й іншими мобільними, десктопними чи вебклієнтами. Цьому сприяє реалізована автентифікація, базована на токенах. Сервіс CVGenerator може використовуватись для представлення наданих йому даних користувача у вигляді документів будь-якої структури, згідно з надісланим клієнтом шаблоном.

РОЗДІЛ 1. ОГЛЯД НАЯВНИХ СИСТЕМ УПРАВЛІННЯ НАВИЧКАМИ

Сучасні системи, пов’язані з веденням обліку навичок, можуть бути призначені як для індивідуального використання, так і для застосування в масштабі підприємств. Зазвичай такі рішення реалізовані на базі хмарних технологій.

Системи обліку навичок відіграють важливу роль у саморозвитку, дозволяючи людям систематизувати уявлення про свої уміння. Це дозволяє оцінити поточний рівень володіння навичками та створити план для подальшого самовдосконалення. Такі системи допомагають контролювати особистий прогрес під час навчання, візуалізувати досягнення та розробляти плани для майбутнього. Ведення запису своїх здобутків сприяє кар’єрному зростанню й може допомогти розширити коло можливостей на ринку праці. Використовуючи платформу для демонстрації своїх умінь, потенційні кандидати мають більші шанси бути поміченими роботодавцями.

Роль систем відстеження професійних навичок на підприємствах полягає в проведенні управління талантами співробітників. Вони допомагають підтримувати й розвивати компетентність працівників і стимулювати поширення нових знань у колективі. Аналізуючи рівень володіння робітниками певних умінь, можна визначити пріоритетні напрямки їх вдосконалення шляхом організації тренінгів і обміну досвідом. Це дозволяє компаніям виявляти загальні прогалини в компетентності штату й укладати рішення щодо набору необхідних кадрів.

1.1. Огляд системи Skill Base

Skill Base [3] – це один із найпопулярніших постачальників рішень в сфері управління талантами. Його пропозиція представлена хмарним програмним забезпеченням, що призначене для управління навичками працівників, узгодження їх розвитку з бізнес-цілями підприємства. Важливою складовою є відстеження прогресу здобуття навичок співробітниками.

Кожен співробітник, зареєстрований у системі, має персональний профіль (рисунок 1.1), який демонструє зріз інформації про його компетентності. Сторінка працівника має вигляд дошки, що містить середні оцінки його навичок, рівень знань за категоріями умінь, список інтересів і статистику показників навчання. Крім того, профіль пропонує підбір користувачів, які мають схожі інтереси й уміння.

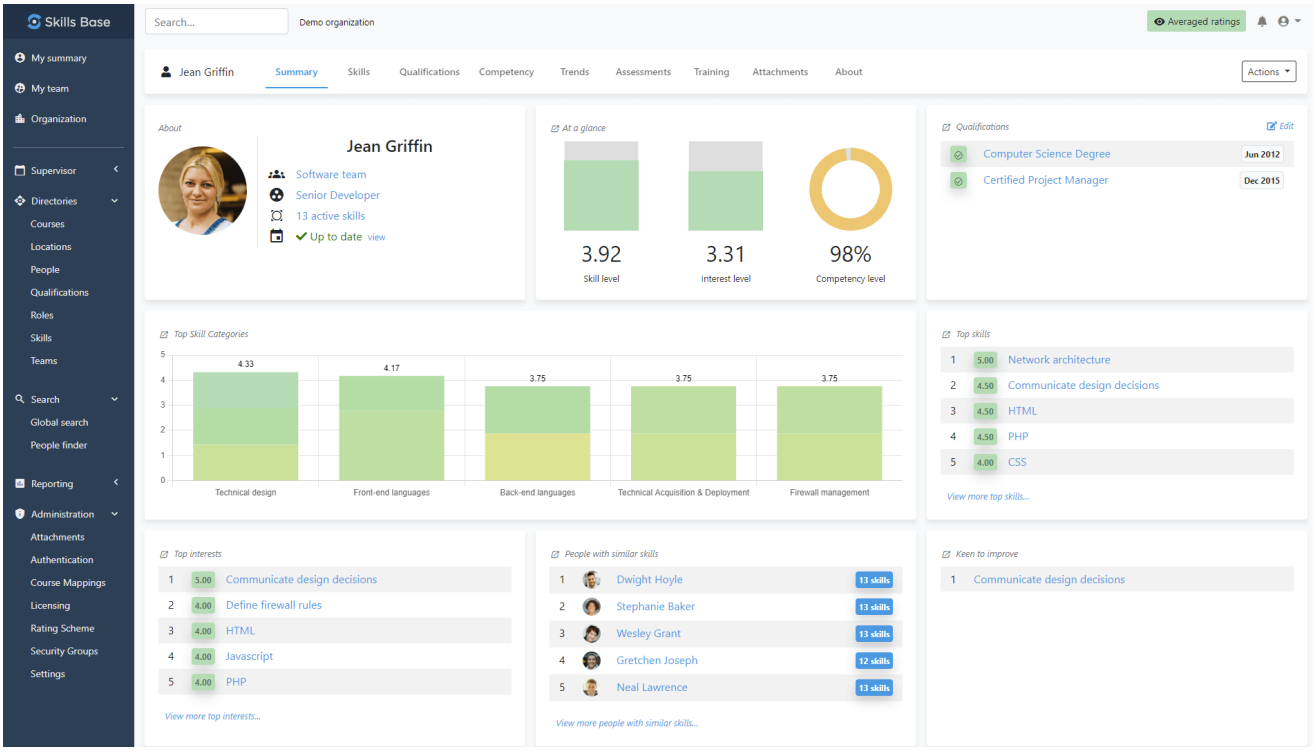


Рисунок 1.1 – Персональний профіль користувача

Адміністратор організації має право управління навичками, навчальними курсами, ролями та командами співробітників. Є можливість додавання й редагування набору професійних навичок, організованих за категоріями та підкатегоріями.

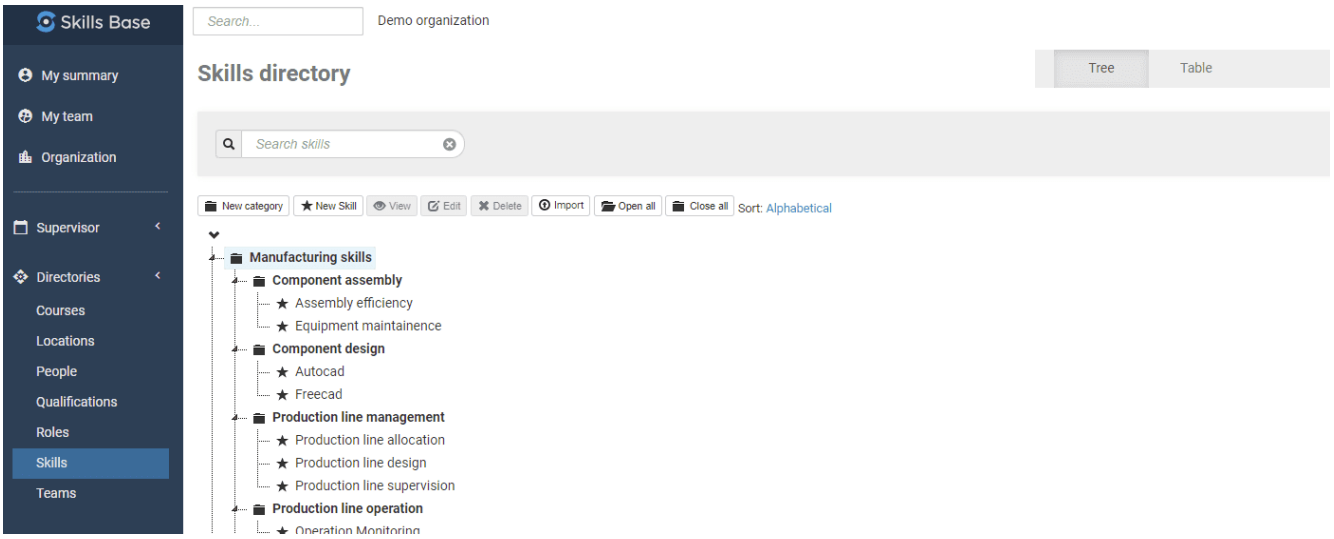


Рисунок 1.2 – Управління навичками у застосунку Skill Base

У рамках організації адміністраторами можуть установлюватися ролі й посади, які співробітники можуть займати в компанії. Створені ролі використовуються в конфігурації різних шляхів кар'єрного зросту в межах підприємства. Для кожної посади можна призначити навички, необхідні для її здобуття. Об'єднавши мапу кар'єрного шляху із доданими навичками, користувачі отримують чіткий план свого розвитку й систему, що його супроводжує. Відповідність співробітника деякій посаді візуалізується у вигляді матриці компетентностей (рисунок 1.3), яка демонструє різницю нинішнього рівня володіння знаннями із необхідним.

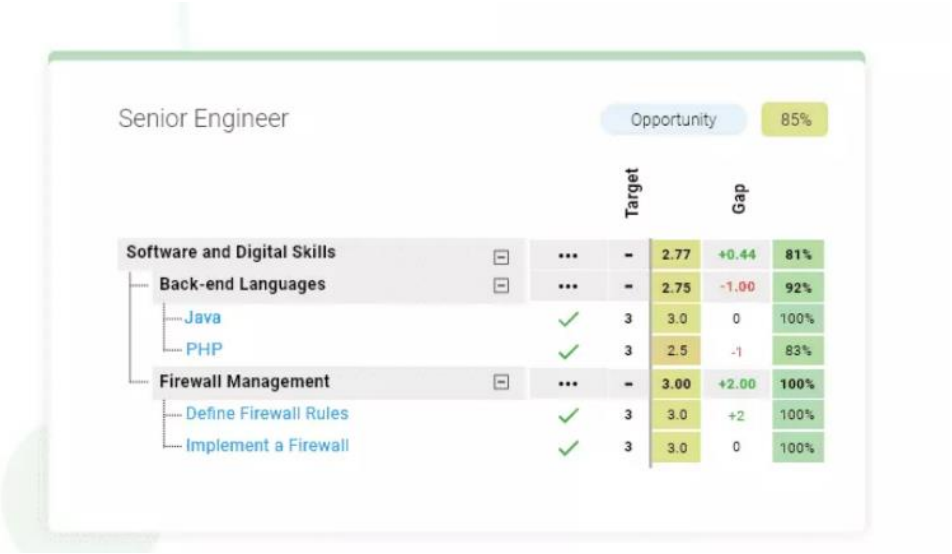


Рисунок 1.3 – Матриця компетентностей

Підбиваючи підсумок огляду платформи Skill Base, необхідно відзначити широкий спектр можливостей, пов’язаних із супроводженням розвитку професійних навичок співробітників. Утім, дане рішення призначення суто для застосування в межах одного підприємства, що суттєво обмежує опції галузевого нетворкінгу користувачів.

1.2. Огляд платформи LinkedIn

LinkedIn [4] – це провідна світова професійна платформа з величезною базою користувачів. Вона пропонує багато можливостей, серед яких пошук вакансій, побудова професійних зв’язків і демонстрація своїх здобутків. LinkedIn дозволяє користувачам створювати профілі під свої потреби, що містять інформацію про їхній досвід роботи, освіту та навички. Завдяки великій і різноманітній базі користувачів, LinkedIn слугує цінним джерелом кандидатів для працедавців.

LinkedIn пропонує можливість гнучкого текстового пошуку (рисунок 1.4). За введеним запитом користувач отримує результат у вигляді списку людей, компаній, вакансій, курсів і загальних публікацій, що містить платформа. Обравши бажану

категорію пошуку, можна налаштувати спеціальні пошукові фільтри, характерні для цієї категорії.

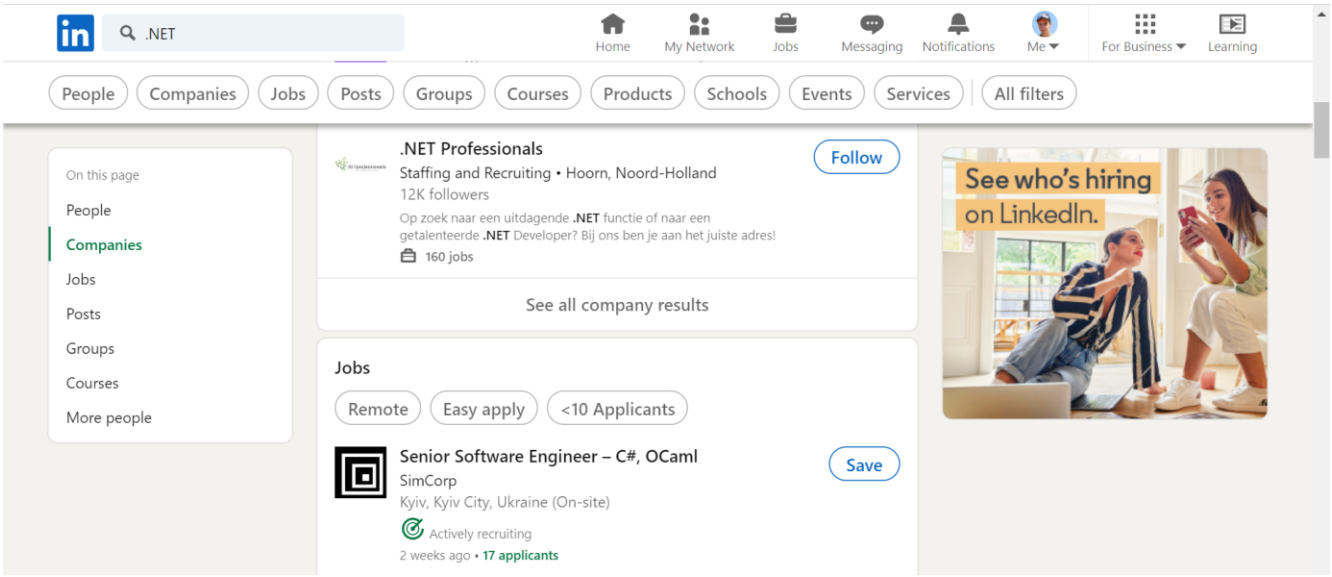
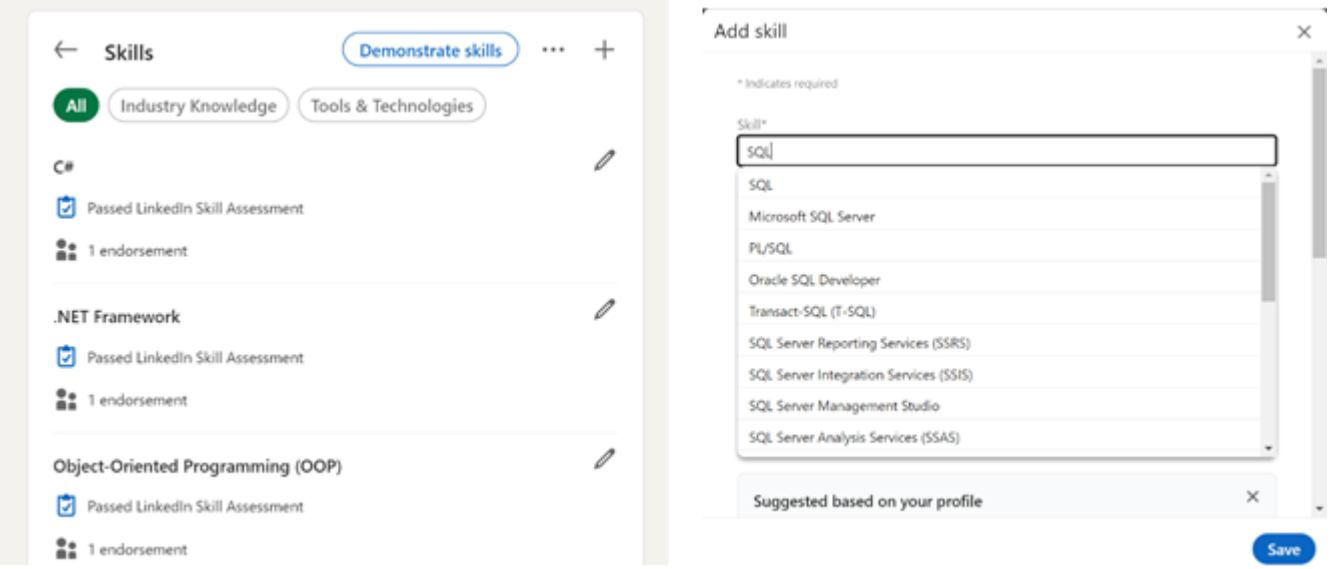


Рисунок 1.4 – Пошук на платформі LinkedIn

Розглядаючи можливості, пов'язані з професійними навичками, слід звернути у вагу на опції з демонстрації своїх умінь на платформі. Профіль користувача LinkedIn містить секцію, створену для показу умінь людини (рисунок 1.5). Для додавання професійного уміння відбувається шляхом простого текстового пошуку. Для підтвердження володіння доданою навичкою існує дві опції:

– отримання підтвердження від інших користувачів;
– підтвердження через проходження оцінювання LinkedIn.

Проте жоден із методів не надає можливості демонстрації загалу якісної оцінки рівня своєї експертизи. Підтвердження від інших користувачів виливається лише у відображання кількості цих підтверджень. Пройшовши оцінювання LinkedIn натомість можна отримати більш детальну оцінку знань. Проте, рівень проходження тесту доступний для перегляду лише користувачеві, що його склав.



а) б)

Рисунок 1.5 – а) Демонстрація навичок у профілі користувача; б) Додавання навички

Підсумовуючи огляд LinkedIn, можна зробити висновок, що головним фокусом платформи є надання можливості побудови мережі професійних зв’язків для спілкування й пошуку роботи. Свою роль у цих процесах відіграють функції демонстрації та оцінки професійних навичок, утім вони не є самодостатнім компонентом системи. LinkedIn заохочує користувачів до пошуку роботи в межах платформи, не надаючи можливості експорту даних користувача, наприклад, у вигляді резюме чи звіту.

1.3. Огляд продукту Zety

Zety [5] є однією з онлайн платформ, що спеціалізуються на наданні можливостей створення й супроводу професійних резюме. Система має вебінтерфейс для створення CV за налаштовуваними шаблонами, які дозволяють людям презентувати свої вміння та досвід. Окрім цього, Zety має рекомендації із застосування найкращих практик написання резюме й пропонує додаткові ресурси з порадами щодо пошуку роботи й кар'єрного зростання.

Наразі Zety має вибір із більш ніж двадцяти шаблонів резюме із можливістю вибору кольорової гами для кожного з них (рисунок 1.6).

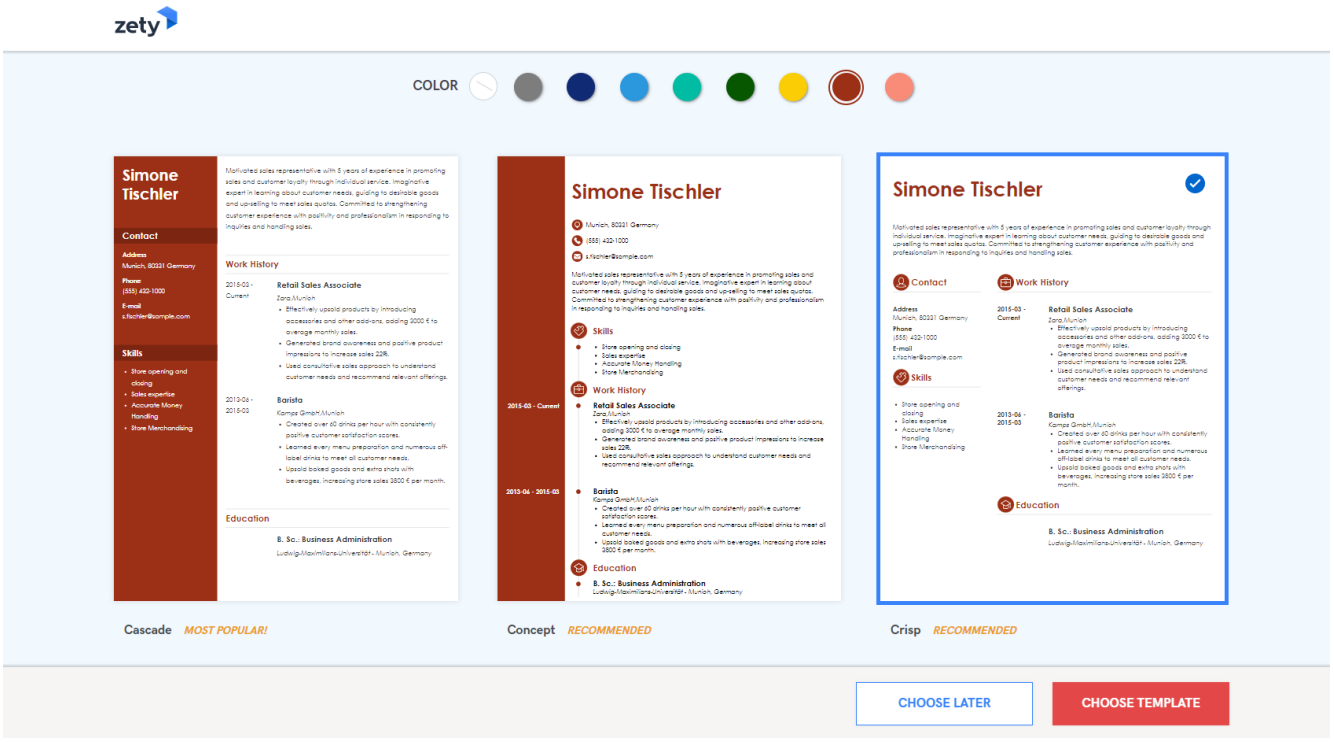


Рисунок 1.6 – Вибір шаблону резюме

Після вибору шаблону користувач має пройти п'ять етапів зі створення резюме (рисунок 1.7).

Етапи наповнення документа даними включають такі пункти:

1. Дані заголовку: ім’я, професія, контакти та локація;
2. Історія роботи;
3. Розділ освіти;
4. Список основних професійних умінь;
5. Короткий опис кандидата.

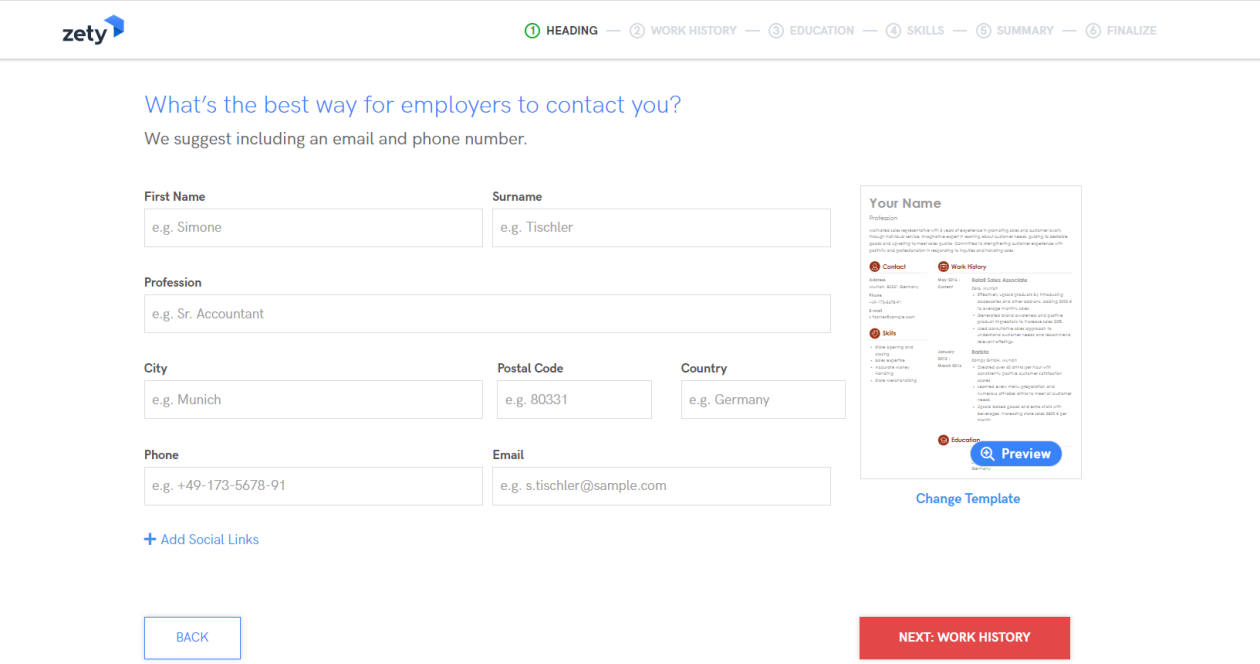


Рисунок 1.7 – Етапи створення резюме

На особливу увагу заслуговує секція із демонстрацією професійних навичок людини (рисунок 1.8). Користувач може додати будь-яке уміння до резюме та вказати свій рівень. Назва навички вводиться в текстове поле, а рівень зазначається кількістю «зірочок» відповідного індикатора. Система містить базу даних із деякими загальними уміннями, які пропонується для вибору.

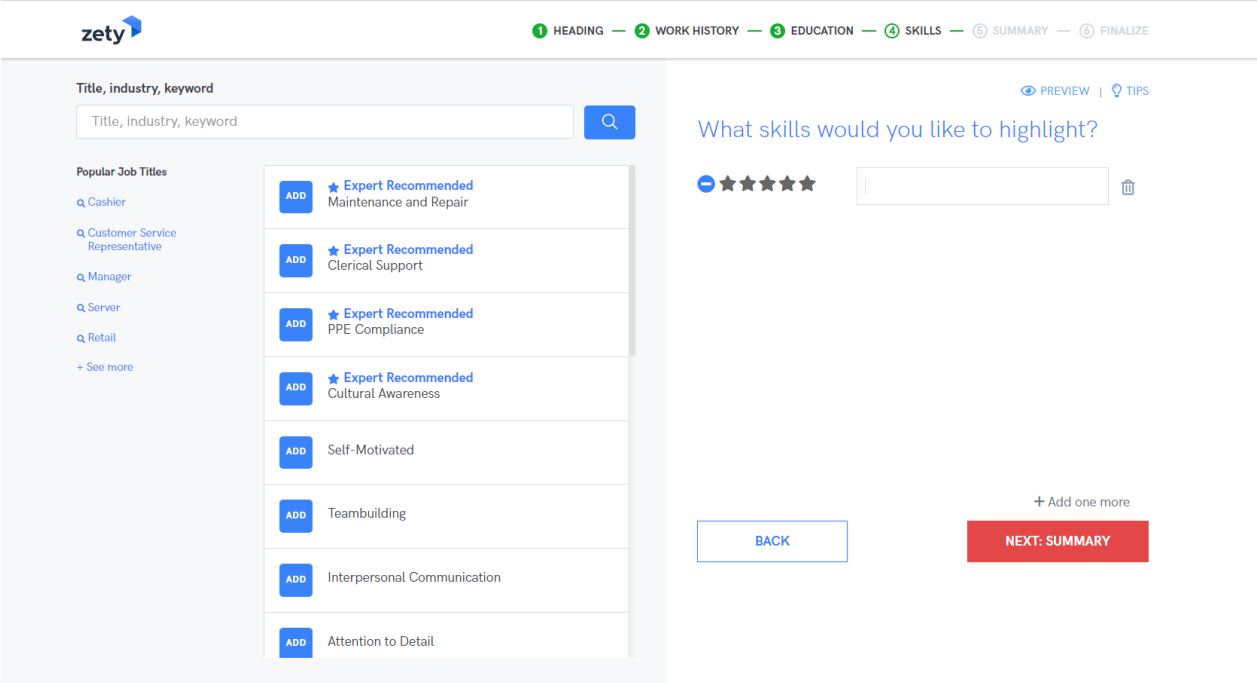


Рисунок 1.8 – Додавання навичок до резюме

Надавши всі необхідні дані, користувач має змогу завантажити резюме у вигляді PDF-файлу, відредагувати його або зберегти. Zety дозволяє зберігати створені резюме на сайті, даючи змогу їх змінювати й використовувати в майбутньому.

Розглянута система призначена для створення й конфігурації резюме й мотиваційних листів. Крім того, Zety зосереджена на допомозі кандидатам у пошуку роботи. Обравши сферу діяльності та бажану позицію, можна скористатися рекомендаціями зі створення успішного резюме. Природно, застосунок спрямований виключно на демонстрацію навичок користувачів, не надаючи опцій їх розвитку.

За результатами огляду можна зробити висновок про доцільність створення єдиної системи для супроводження саморозвитку. Надаючи можливості комерційних рішень управляння талантами широкому загалу користувачів, така система має споріднити процес поступового здобуття навичок із їх демонстрацією потенційним роботодавцям.

РОЗДІЛ 2. ВЛАСТИВОСТІ АРХІТЕКТУРИ МІКРОСЕРВІСНИХ СИСТЕМ

Розподілена мікросервісна архітектура програмних застосунків – це база проєктування багатьох сучасних розширюваних застосунків [6]. Вона полягає у розробці системи, що складається з декількох окремих самодостатніх сервісів. Кожен із них покликаний розв'язувати свою задачу й може розроблятися та підтримуватися незалежно від інших. Працюючи в рамках єдиної системи, вони дозволяють вирішувати складні багатокомпонентні бізнес-завдання.

Порівняно з монолітними застосунками, розподілені системи мають ряд функціональних переваг, проте й породжують додаткові проблеми та задачі. Для їх вирішення створено низку шаблонів і принципів, які дозволяють створювати гнучкі розширювані системи.

2.1. Переваги мікросервісної архітектури

Мікросервісна архітектура – це підхід до розробки програмного забезпечення, який передбачає розбиття складних застосунків на менші, незалежні сервіси. Ці сервіси можуть взаємодіяти один з одним через інтерфейси API та розроблятися, розгортатися й масштабуватися незалежно один від одного. Такий підхід забезпечує більшу гнучкість у розробці програмного забезпечення, що полегшує управління великими та складними системами.

У книзі [6] автор визначає такі ключові переваги мікросервісної архітектури:

- командна автономність;
- сервісна автономність;
- масштабованість;
- ізоляція несправностей;

– автономність даних.

Командна автономність. Архітектура мікросервісів надає значні переваги в автономності команд, що працюють над проєктом. Розбиваючи складні застосунки на менші, незалежні сервіси, групи розробників можуть працювати над призначеними їм проблемними областями окремо від інших команд. Це дозволяє кожній групі розробляти компоненти програми у власному темпі, що в загальній перспективі дає компаніям конкурентну перевагу за рахунок зменшення часу на випуску продукту.

Мікросервісна архітектура також дозволяє використовувати відмінні технології для програмування кожного сервісу, включно з мовами програмування, способом збереження даних і тестування. Це забезпечує гнучкість у виборі інструменту для реалізації логіки кожного сервісу. Кожна з команд має можливість зосередитись на реалізації й тестуванні свого компонента, використовуючи найбільш ефективні технології. Не є обов’язковим знання внутрішньої структури сервісів, які розробляються іншими командами. Зв'язок між ними здійснюється, зокрема, через кінцеві точки API. Взаємодія між командами може проводитися на рівні проєктування та узгодження контракту – інтерфейсу, який кожен сервіс пропонує для взаємодії з іншими.

Можливість розділення відповідальності за компоненти системи між різними командами також полегшує імплементацію сучасних гнучких методів розробки. Agile-команда є крос-функціональною групою, що зазвичай складається з 10 або менше розробників, які мають необхідні уміння для визначення, створення, тестування й видачі цінності продукту [7].

Сервісна автономність. Ця риса передбачає розділення загальної проблеми на рівні сервісів, що, зокрема, задовольняє принципу єдиної відповідальності SOLID. «Жоден мікросервіс не повинен мати більше однієї причини для зміни. Наприклад, мікросервіс «Управління замовленнями» не повинен також складатися

з бізнес-логіки для управління акаунтами. Маючи мікросервіс, присвячений конкретним бізнес-процесам, сервіси можуть розвиватися незалежно» [6]. Такий підхід дозволяє компонентам системи існувати окремо, покращуючи їхню індивідуальну стійкість до помилок та доступність. Слабкий зв'язок (loose coupling) між ними означає, що зміни в одних сервісах мають менший вплив на інші, або не мають жодного, якщо стосуються виключно внутрішньої реалізації. Створення повторно використовуваних компонентів покращує якість кожного компонента, оскільки спільна база коду ефективніше піддається виявленню помилок та слабких місць [8].

Масштабованість. Для великих комерційних застосунків масштабованість є найбільшою перевагою використання розподілених архітектур. Виконуючи масштабування, застосовують два основні підходи: горизонтальне й вертикальне масштабування (рисунок 2.1) Для поліпшення роботи монолітних застосунків застосовують вертикальне масштабування. Зазвичай воно досягається шляхом додавання додаткових ресурсів до поточних серверів, щоб збільшити потужність ЕОМ, на якій виконується програма. Це дозволяє збільшити навантаження на сервер [9]. Мікросервіси, у свою чергу, можна масштабувати горизонтально, додаючи більше екземплярів на декілька серверів. Такий підхід забезпечує кращу ізоляцію несправностей і дозволяє сервісам працювати незалежно.

Крім того, розподілена архітектура дозволяє різним сервісам використовувати сервери різної потужності та конфігурації. Наприклад, один може потребувати більших ресурсів процесора, в той час, як іншому може знадобитися більший обсяг пам’яті. Доцільне використання ресурсів серверів оптимізує продуктивність і масштабованість застосунків.

Масштабування дозволяє ефективно використовувати фінанси замовника. Кожен сервіс може використовувати виключно необхідні йому засоби, запобігаючи витрачанню коштів на підтримку надмірних ресурсів.

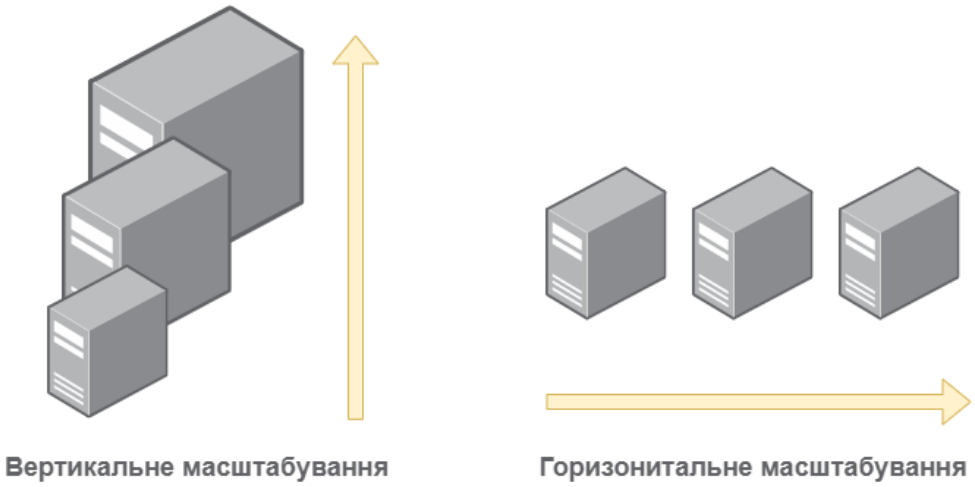


Рисунок 2.1 – Підходи до масштабування систем

Ізоляція несправностей. Ця характерна риса мікросервісної архітектури допомагає запобігти впливу збоїв в одному сервісі на роботу інших. Коли збій відбувається в монолітному застосунку, він ризикує вивести з ладу всю систему. На противагу цьому, мікросервісна архітектура дає змогу логічно ізолювати сервіси так, щоб максимально обмежити потенційний вплив проблеми одного сервісу на інші.

Однак це не означає, що розподілена система мікросервісів повністю застрахована від несправностей. Вихід із ладу одного компоненту так чи інакше зупиняє виконання деякої частини функціонала системи. В таких випадках цей сервіс може бути перезапущений. Оркестратори, такі як Kubernetes, допомагають керувати цим процесом і запобігають виведенню з ладу системи після помилки в окремому сервісі.

Автономність даних. Ця перевага полягає в тому, що зміни в організації даних деякого компонента системи є ізольованими від інших. Так само й при необхідності випуску нової частини функціонала може бути створений окремий

сервіс із власною БД. Інше застосування – використання різних технологій зберігання даних для різних сервісів.

2.2. Проблеми мікросервісної архітектури

Описані риси мікросервісів свідчать про їх очевидну перевагу над монолітною архітектурою в підтримуваності, розширюваності й надійності. Утім, реалізація такого підходу породжує також і додаткові проблеми, на які необхідно зважати при проєктуванні системи.

Складність. Застосування такого підходу виливається в більшу складність системи. Кожен із сервісів може бути простим, проте спільна організація багатьох компонентів, що функціонують незалежно, є складнішою. В розподіленій системі утворюється значна кількість комплексних залежностей на її різних рівнях, що ускладнює розробку та тестування. Складність системи створює додаткові виклики в її управлінні. Розгортання багаторівневого застосунку вимагає більших зусиль, ніж проведення того самого процесу для монолітного застосунку.

Синхронізація даних. Важливою є проблема підтримки узгодженості даних між сервісами. Кожен із них може мати власне сховище даних, тож необхідно забезпечувати послідовну зміну даних і запобігати використання застарілої інформації. Ця проблема має вирішуватися комунікацією сервісів між собою. Зв’язок між сервісами, зокрема, з метою синхронізації даних змушує покладатися на доступність і швидкість мережі. Залежність системи між мережі також є проблемою, адже її потенційні затримки можуть суттєво знижувати ефективність системи, компоненти якої активно обмінюються повідомленнями.

Недоліки розрізнених технологій. Слід також згадати про технологічні недоліки використання розподіленої архітектури. Розбіжність між технологіями, які використовуються для розробки компонентів системи, у перспективі може

ускладнити її підтримку. Це особливо проявляється при реалізації міжсервісних складових, таких як логування та моніторинг. Команди є обмеженими у виборі інструментів для розв'язання таких задач і мають узгоджувати їх застосування з іншими, що вимагає більше часу на комунікацію та дослідження сумісності технологій. Крім того, розробка комплексної архітектури вимагає більшої компетентності команди в цілому.

Таким чином, на етапі проєктування системи важливо зважати як на переваги, так і на недоліки розподіленої архітектури. Слід брати до уваги такі аспекти, як складність продукту, що розробляється, перспективи його підтримки та розширення, компетентність команди, обмеження в часі та доступних ресурсах.

2.3. Стандартні підходи та шаблони розробки

Окрім загальної структури системи, важливе значення має архітектура її індивідуальних компонентів – мікросервісів. Вони, як і система загалом, мають бути розширюваними й підтримуваними. Для їх реалізації застосовують низку принципів і шаблонних підходів, які запобігають виникненню типових проблем під час майбутньої підтримки програм.

2.3.1. Паттерн CQRS

CQRS – це шаблон проєктування, який полягає у розділенні операцій читання та запису в моделі даних застосунку. Використовуючи CQRS, можна оптимізувати продуктивність, масштабованість, безпеку і можливість обслуговування системи. Такий підхід обумовлений тим, що більшість запитів до сервера, зазвичай, припадає на операції читання [10]. Операції читання не модифікують дані й можуть бути логічно або фізично відокремлені від операцій зміни даних.

Базова реалізація паттерну CQRS, запропонована в статті [11], наведена на рис. 2.2. Вона полягає у створенні двох моделей доступу до даних у програмі – запитів (Query Model) і команд (Command Model). Рисунок демонструє застосування шаблону до системи з єдиною базою даних. Проте, поширеним є використання окремих сховищ для читання й зміни стану. Таке рішення стає в нагоді при необхідності поліпшити доступність системи при виконанні одного класу операцій. Зазвичай цим типом є читання. Існує можливість удатися до горизонтального масштабування сховища даних, призначеного для читання, створивши додаткові екземпляри БД.

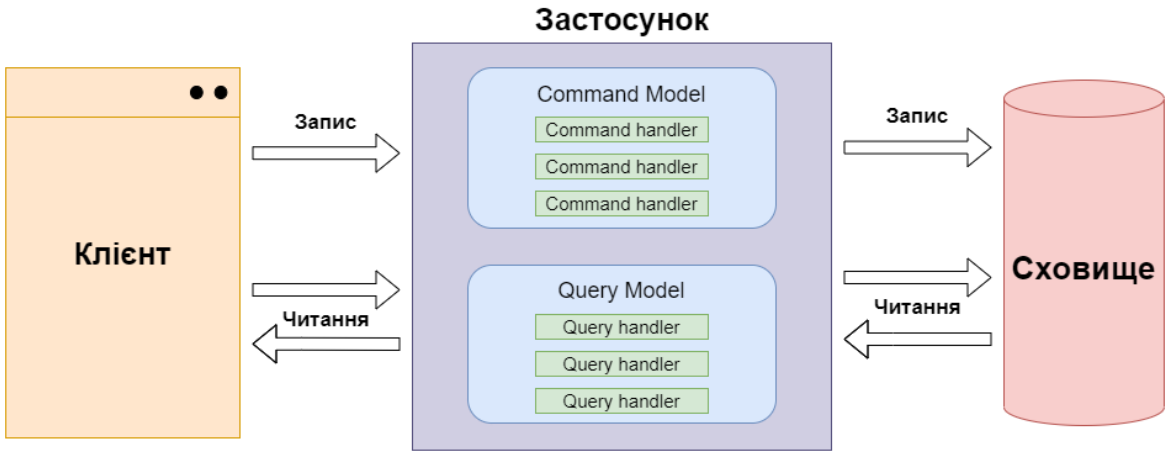


Рисунок 2.2 – Реалізація CQRS

Запити представляють об’єкти, які використовуються для читання даних. Вони містять методи для отримання даних, можуть мати параметри, використовувані для звернення до БД, сортування отриманої вибірки. Запити мають опрацьовуватися обробниками запитів (Query Handlers), що повертають шукані значення.

Команди є об’єктами, призначеними для виконання операцій зі зміни стану програми. Такими операціями є Insert, Update та Delete. На відміну від запитів, обробники команд не повертають жодних даних, утім можуть повідомляти про

успішність виконання команди. Для них спеціально відведені обробники команд (Command Handlers).

2.3.2. Багаторівневі архітектури

Традиційно ефективною практикою при розробці монолітних сервісів вважається дотримування принципу поділу інтересів (Separation of Concerns). Він полягає в розділенні застосунку на декілька шарів, пов’язаних між собою. Організований за таким принципом код простіше підтримувати та розвивати.

Одним зі способів дотримання принципу SoC є використання **багатошарової архітектури** (рисунок 2.3). Зазвичай цей підхід передбачає наявність трьох рівнів:

- Рівень даних (DAL), який відповідає за доступ до сховища даних;
- Рівень бізнес-логіки (BLL), що містить реалізацію функціональної логіки застосунку;
- Презентаційний рівень (PL), котрий представляє зовнішній інтерфейс програми: UI, API тощо.

Застосунок будується так, що його «верхні» шари залежать від «нижніх». Таким чином спосіб збереження даних природно не залежить від бізнес-логіки, яка, своєю чергою, незалежна від реалізації зовнішнього інтерфейсу. Усі рівні можуть звертатися до умовного інфраструктурного компоненту (Infrastructure), що містить допоміжні універсальні функції.

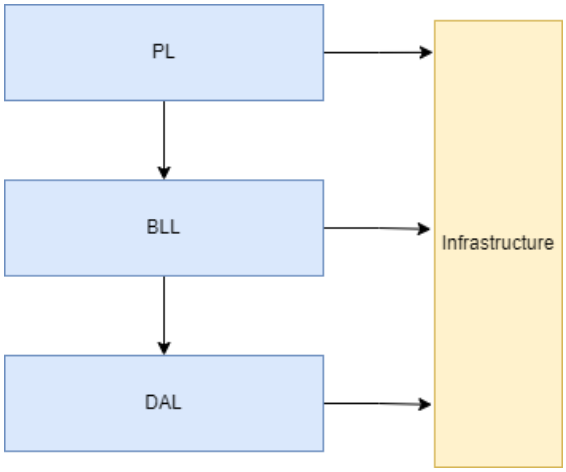


Рисунок 2.3 – Трирівнева архітектура

Недоліком описаної архітектури є залежність презентаційного рівня від рівня даних. Хоч вони й не пов’язані напряму, існує транзитивна залежність через рівень бізнес-логіки. Через це зміни шару DAL автоматично означають ризик оновлення рівня PL, що помітно ускладнює модифікацію програми. Для вирішення цією проблеми Jeffrey Palermo у 2008 році був запропонований новий підхід – так звана «цибулева» архітектура (Onion Architecture) [12].

На рис. 2.4 зображена діаграма Onion-архітектури. У її ядрі – модель домену програми, що є представленням центральних правил та понять предметної області системи. Ключовим є те, що доменна модель є лише абстракцією, яка не залежить від способу зберігання даних. БД у цій архітектурі винесена у вигляді зовнішньої інфраструктурної залежності.

Onion-архітектура тісно пов’язана із принципом інверсії залежностей (Dependency Inversion) – одним із принципів SOLID, запропонованих Робертом Мартіном [13]. Він полягає в тому, що модулі вищих рівнів мають бути незалежними від модулів нижніх рівнів. Усі модулі повинні залежати від абстракцій. Дійсно, нижні шари Onion-архітектури не залежать від верхніх і всі вони залежать від абстракції доменної моделі.

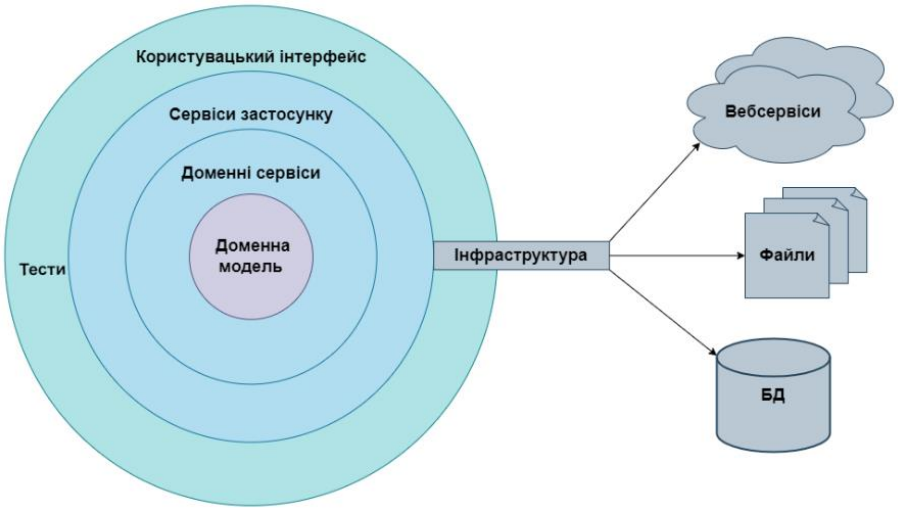


Рисунок 2.4 – Onion архітектура

Іншим популярним підходом проєктування застосунків є «шестикутна» архітектура, або архітектура портів та адаптерів (рисунок 2.5). Як і «цибулева», вона спирається на принцип DI в побудові структури програми. Утім, головною особливістю цієї архітектури є відкритість системи до використання різних шляхів зберігання даних і організації зовнішнього інтерфейсу. Їхні реалізації можуть бути замінені безперешкодно.

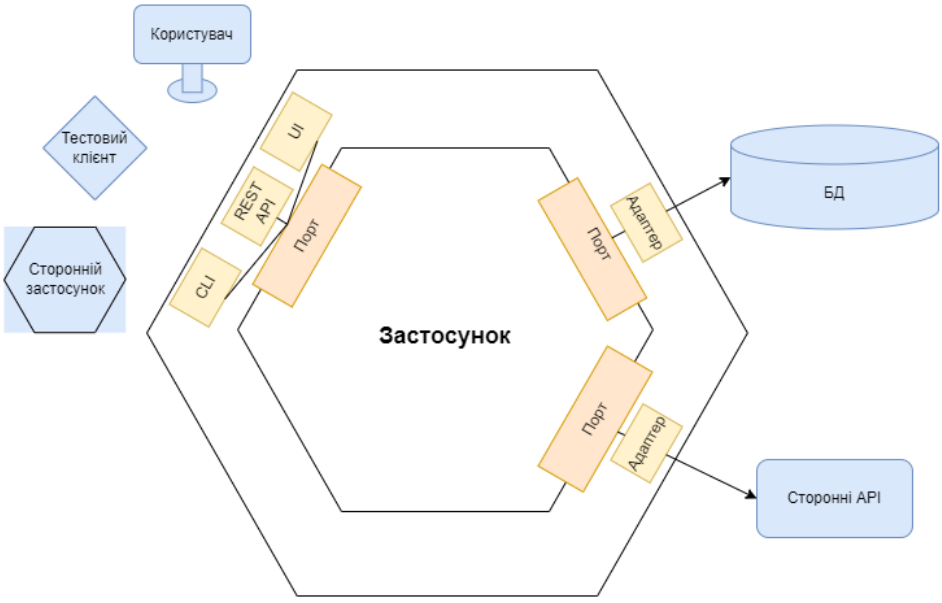


Рисунок 2.5 – Архітектура портів і адаптерів

У статті [14] запропоноване наступне означення ключових компонентів архітектури портів та адаптерів:

- Ядро – логіка застосунку. Може мати різну реалізацію, проте, так чи інакше, базується на абстракції домену;
- Порти – інтерфейси, якими застосунок послуговується для доступу до даних та їх презентації. Своєю чергою, поділяються на Driver Ports – інтерфейси, які визначають API для взаємодії з програмою, і Driven Ports, що стандартизують комунікацію із БД.
- Адаптери є реалізацією інтерфейсів, представлених портами. Система може мати необмежену кількість адаптерів, які реалізують різні способи взаємодії з даними і зовнішніми ресурсами.
- Інфраструктура системи, представлена її конфігурацією та зовнішніми залежностями.

2.3.3. Розробка SPA-застосунків

Традиційним підходом до розробки вебзастосунків вважається створення сайтів, які складаються з декількох сторінок [15].

Користування таким вебсайтом передбачає викачування нової сторінки замість старої щоразу, коли користувач відкриває іншу область застосунку. Необхідність багаторазової передачі сервером повної HTML-сторінки здатне сповільнити в клієнта із застосунком. Проте, такі вебсайти є кращими кандидатами для SEO-оптимізації [16].

Більш сучасним є підхід проєктування SPA – застосунків, що складається з єдиної сторінки, вміст якої динамічно оновлюється. Цей підхід зменшує навантаження на мережу й підвищує швидкодію вебсайту. Він став можливий завдяки розвитку можливостей технологій JavaScript і Ajax у сфері виконання

асинхронних запитів на сервер і оновлення структури вебсторінки. Створення SPA-застосунків підтримується і сучасними вебфреймворками, такими як Angular і React. Схематичне порівняння багато- й односторінкових вебдодатків наведено на рис. 2.6.

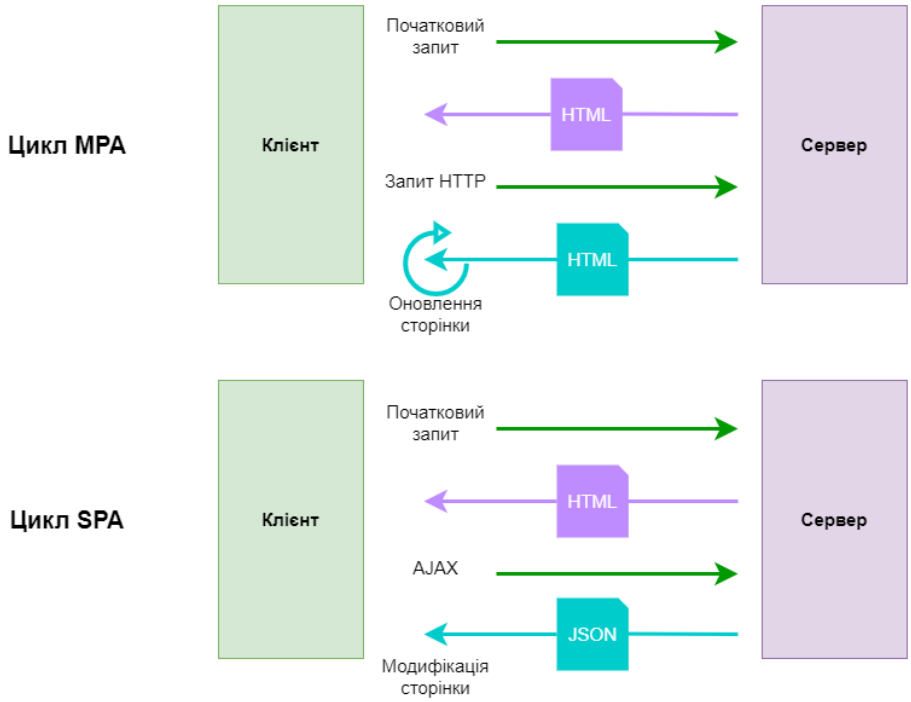


Рисунок 2.6 – Порівняння MPA і SPA застосунків

РОЗДІЛ 3. ОГЛЯД РОЗРОБЛЕНОЇ СИСТЕМИ

Можливості розробленої системи зосереджені навколо роботи з професійними вміннями користувачів. Вебінтерфейс дозволяє здійснювати пошук і оцінку навичок, їх експорт у вигляді резюме, а також перегляд профілів користувачів. Функціональні можливості застосунку зображені на діаграмі прецедентів у додатку А. Продемонструємо головні функції застосунку.

3.1. Робота з професійними навичками

Ключовим компонентом домену системи є сутність Skill (уміння). Застосунок має окрему сторінку, призначену для перегляду та додавання умінь (рисунок 3.1). Навички організовані у вигляді деревоподібної структури, дозволяючи створювати нові уміння на будь-якій глибині.

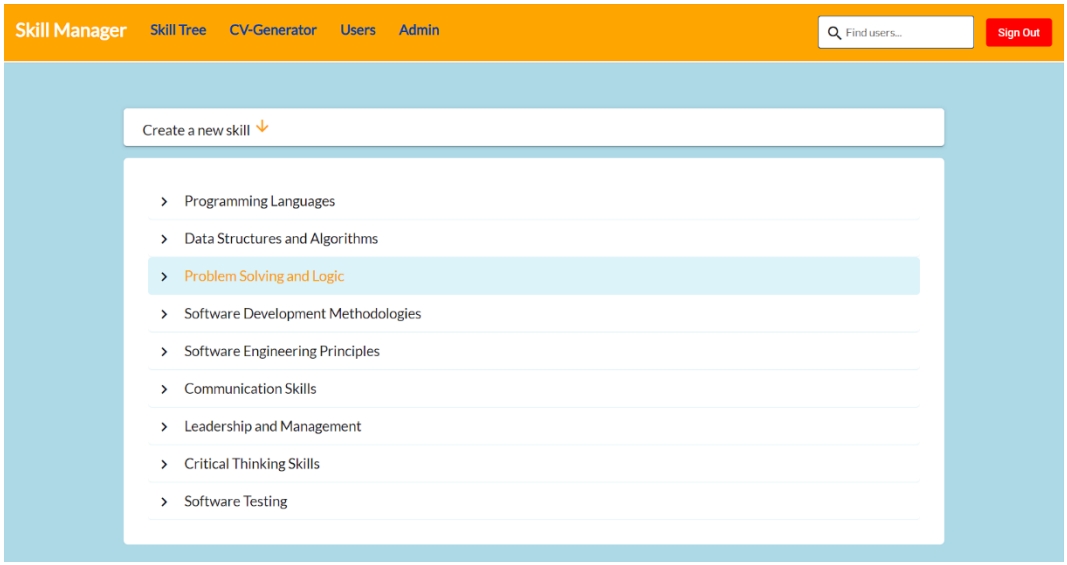


Рисунок 3.1 – Сторінка «Skills»

Сторінка містить форму для додавання нових умінь (рисунок 3.2). Вона має поля для введення назви й опису уміння. Батьківську навичку можна обрати, натиснувши на відповідний елемент списку.

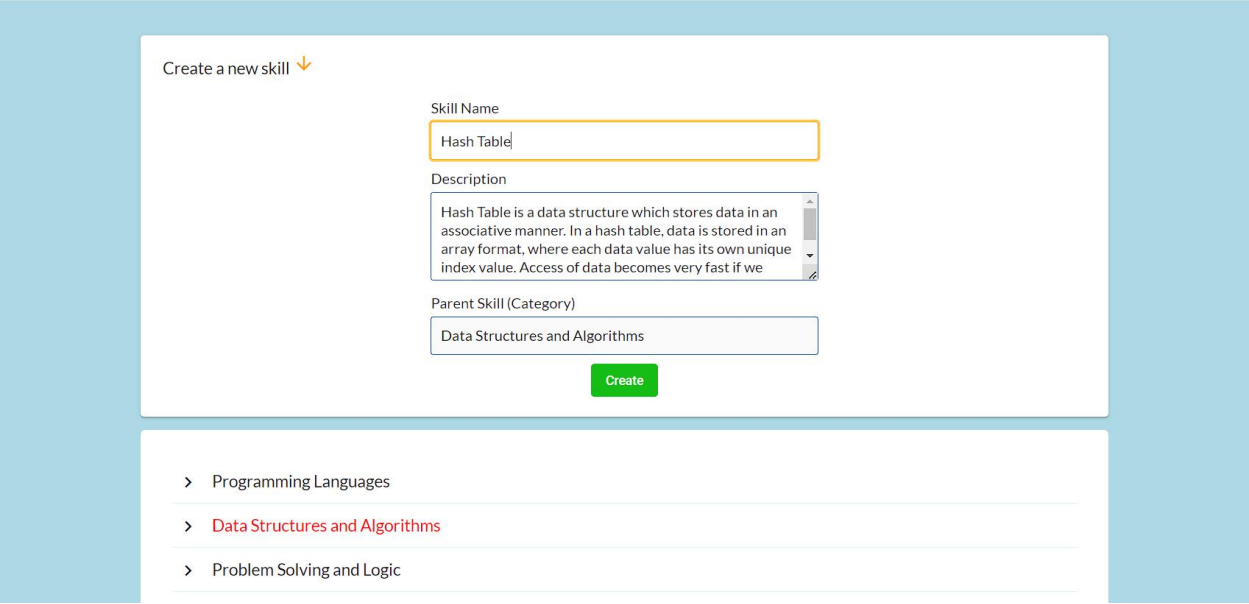


Рисунок 3.2 – Сторінка «Skills» із відкритою формою

Навички, для яких не призначено батьківське уміння, є категоріями й не можуть бути оціненими користувачем. Для дочірніх навичок користувач може зберегти рівень володіння ними за допомогою індикатора (рисунок 3.3). Оцінка передбачає вибір одного з чотирьох рівнів:

- Entry – початковий, людина знайома з даною темою;
- Intermediate – людина демонструє вільне володіння навичкою;
- High – рівень знань можна оцінити як професійний;
- Advanced – людина володіє умінням на просунутому рівні.

При наведенні курсора на показник над ним відображається назва підсвіченого рівня. Окрім того, оцінка візуалізована кольоровим індикатором. Для збереження та зміни оцінки достатньо натиснути на обраний елемент індикатора. Зміни зберігаються в системі автоматично.

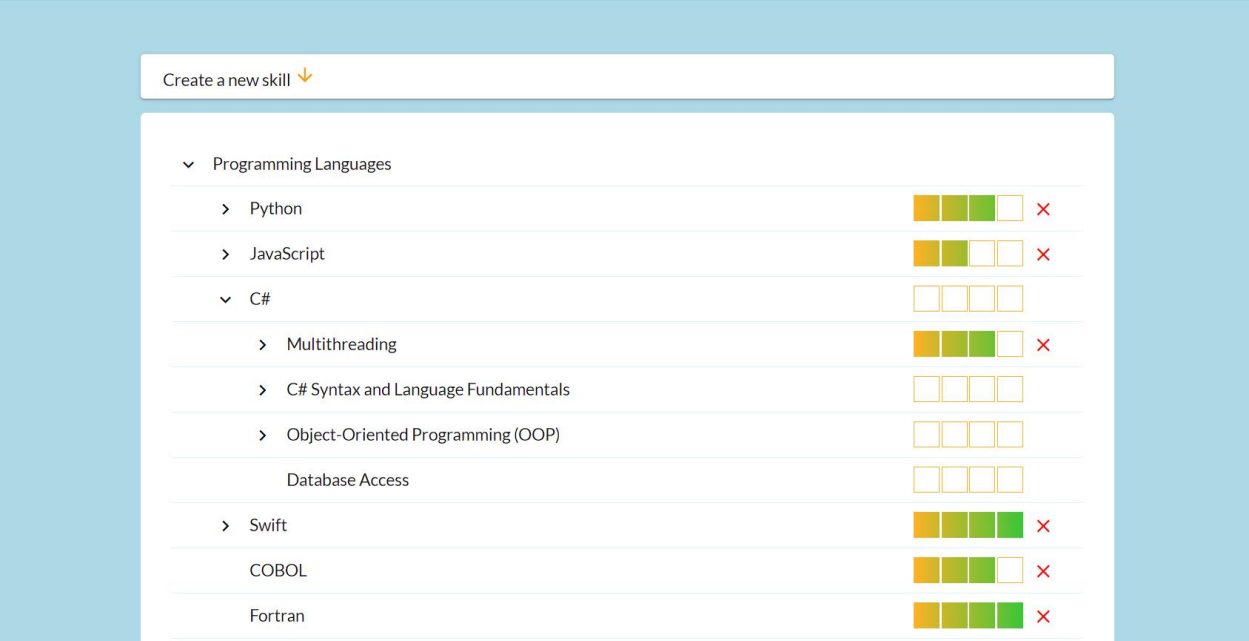


Рисунок 3.3 – Оцінка рівня володіння навичками

Кожен елемент списку є посиланням на сторінку, присвячену певній навичці. Наприклад, рисунок 3.4 демонструє сторінку уміння C#. Вона розділена на дві половини. Ліва частина містить представлення ієрархії умінь. Біля обраної навички показано кількість користувачів, що додали її до свого профіля. Цей елемент також є посиланням на список відповідної вибірки користувачів (рисунок 3.6). Права частина включає індикатор рівня володіння умінням, його опис і кнопку для виклику форми створення дочірньої навички. Текстова частина сторінки може слугувати для збереження опису уміння, посилань на корисні ресурси й онлайн-курси, а також містити будь-які інші релевантні матеріали.

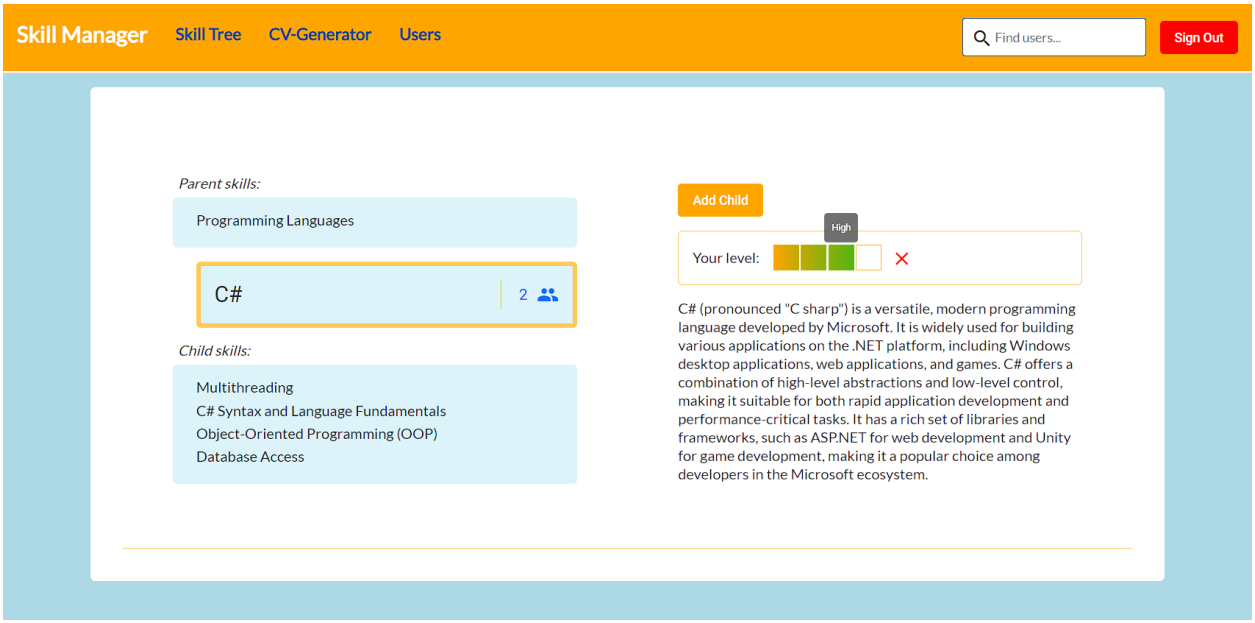


Рисунок 3.4 – Сторінка інформації про навичку

Користувач може створити власну навичку, яка буде доступна для перегляду лише йому та адміністраторам. Це дозволяє використовувати застосунок для супроводження власного навчання та саморозвитку. Людина може створити будь-які навички, які планує здобути, а також додати вже набуті, якщо таких немає в системі. Надалі ці уміння зберігатимуться і в профілі користувача, але не будуть видимі для інших. Утім, творець навичок зможе включити їх до власного експортованого резюме. Рисунок 3.5 зображує сторінку уміння, створеного користувачем, із відкритою формою для додавання нової навички. На сторінці з’являється кнопка Delete, надаючи можливість видалити новостворене уміння.

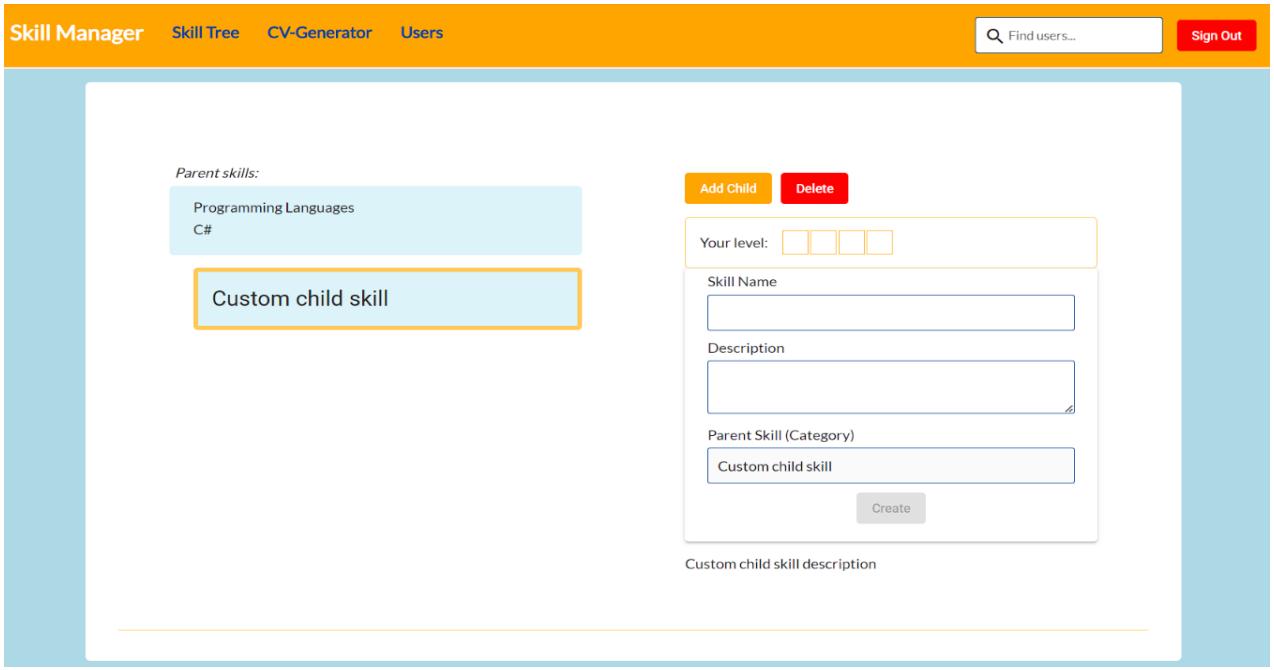


Рисунок 3.5 – Сторінка створеного користувачем уміння

3.2. Робота з користувачами

Будь-який користувач застосунку може здійснити пошук інших користувачів системи та переглянути їх профіль, оцінити їхні навички. Сторінка пошуку (рисунок 3.6) містить список його результатів і поле для введення пошукового запиту. Перейшовши до списку зі сторінки уміння (рисунок 3.4), можна переглянути лише користувачів, які цим умінням володіють.

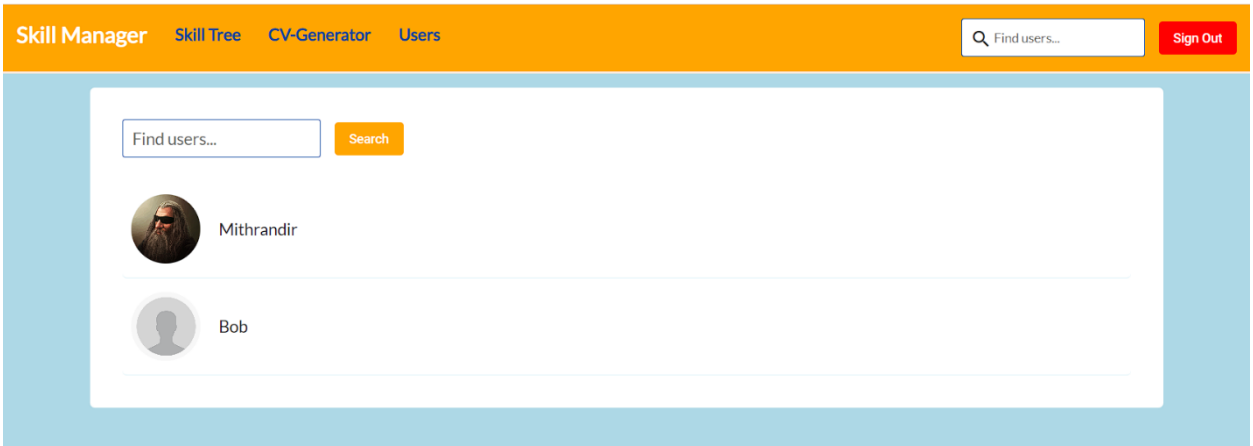


Рисунок 3.6 – Вибірка користувачів за умінням

Публічно доступний профіль (рисунок 3.7) містить дані користувача, такі як його ім’я, посада, зображення, спосіб зв’язку й секцію з короткою інформацію про людину. Вкладка «Additional info» відведена для перегляду додаткової інформації про користувача.

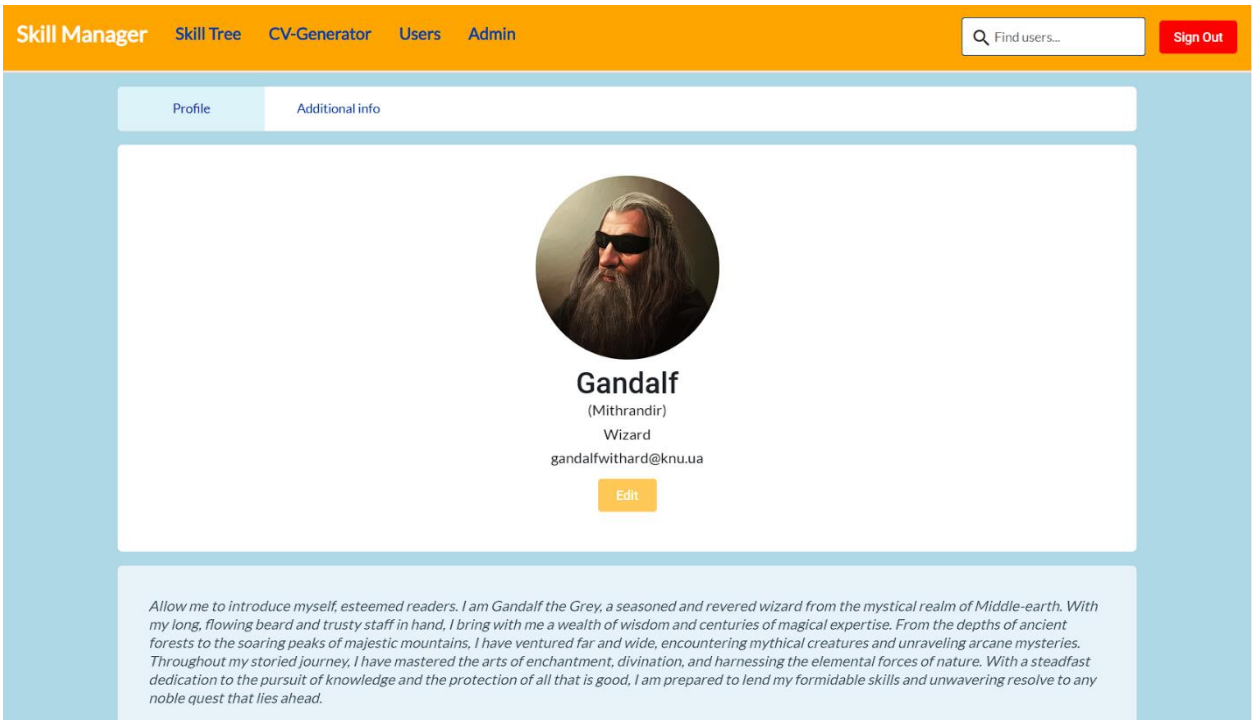


Рисунок 3.7 – Публічний профіль користувача

Додаткова інформація передбачає два розділи: про освіту й досвід роботи. Кожен з розділів представлений списком із пунктів і їх коротким описом (рисунок 3.8).

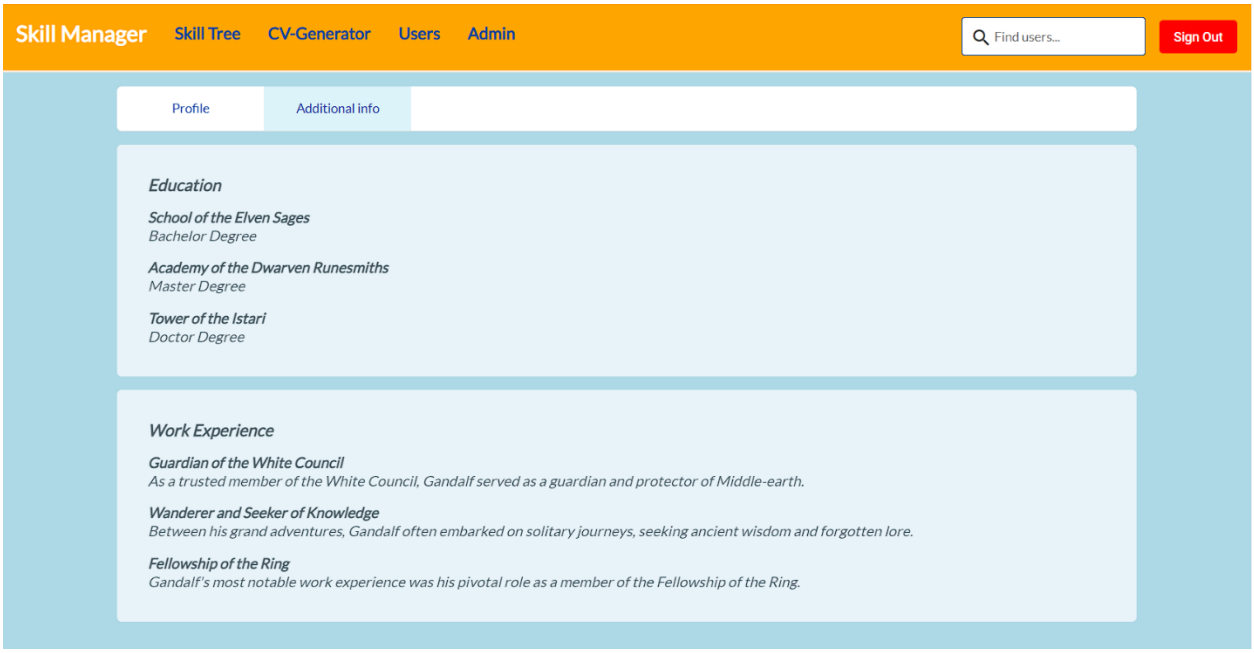


Рисунок 3.8 – Додаткова інформація профіля

У профілі користувача міститься й список умінь, якими він володіє (рисунок 3.9). Це альтернативне представлення списку, де навички згруповані за категоріями. Окрім власної оцінки, тут наявне середнє значення оцінок від інших користувачів і кількість цих оцінок. Перейшовши в профіль іншого користувача, можна дати оцінку його вмінням.

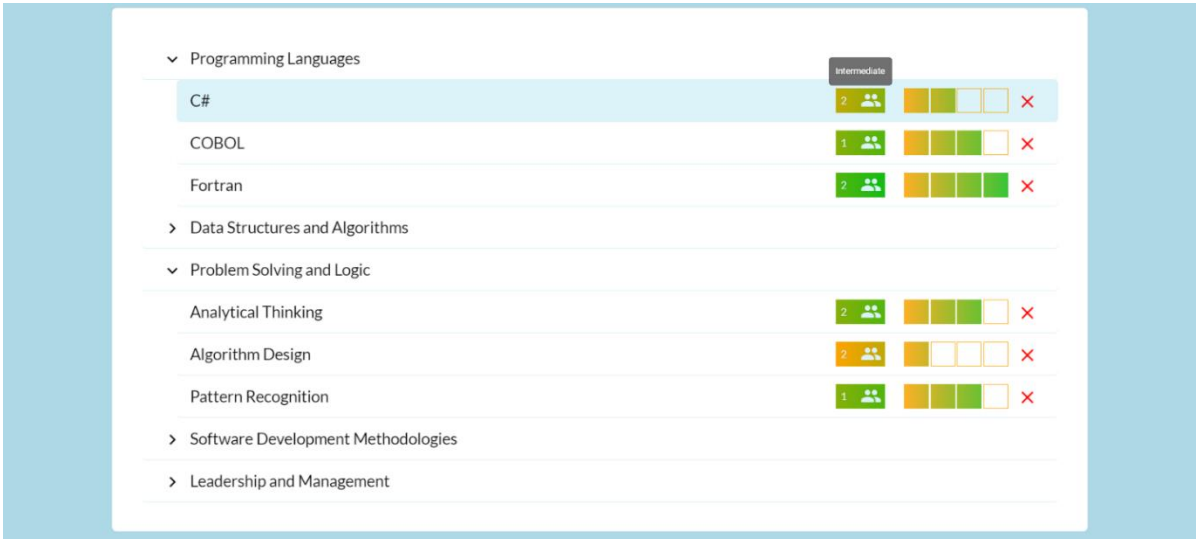


Рисунок 3.9 – Список умінь в профілі користувача

Кнопка Edit в профілі користувача (рисунок 3.7) призначена для переходу на сторінку редагування даних профіля (рисунок 2.10). На цій сторінці можна змінити будь-яку інформацію, представлену в профілі. Дані, збережені на цій сторінці, можуть бути використанні при генерування резюме. Проте, включення цих компонентів у документ не є обов’язковим.

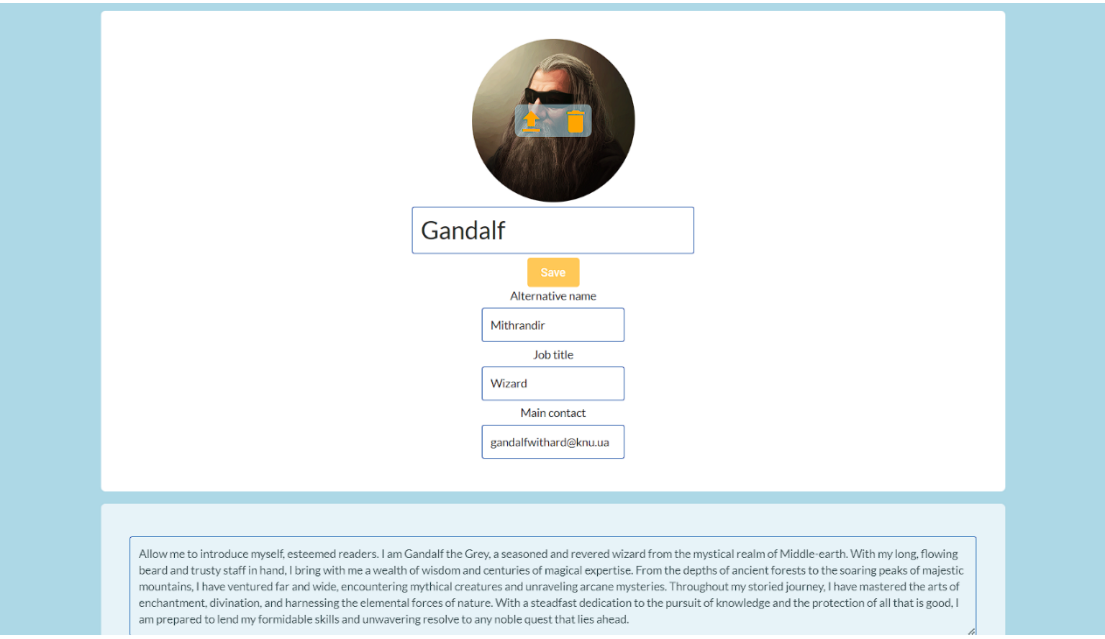


Рисунок 3.10 – Редагування даних профілю

Пункти в історії освіти й роботи користувача можуть видалятися, додаватися та редагуватися в індивідуальному порядку (рисунок 3.11). Тут користувач може додати будь-які пункти, які вважає необхідними. В якості досвіду роботи можуть виступити як займані посади, так і волонтерська робота чи особисті проєкти. Етапи освіти призначені для зберігання історії пройдених навчальних закладів людини, завершених курсів і сертифікацій.

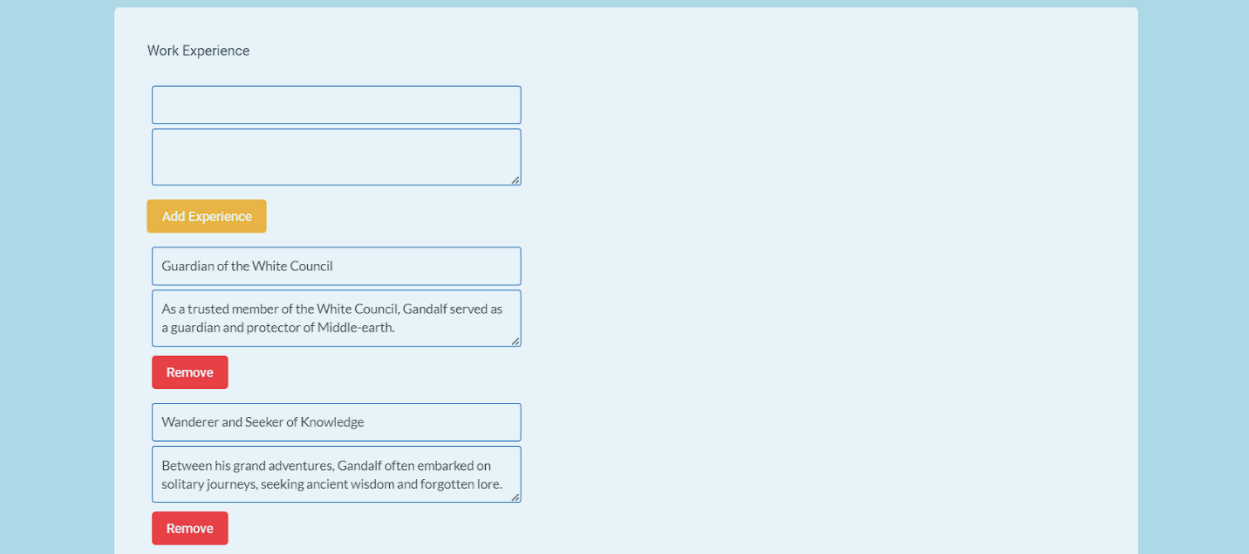


Рисунок 3.11. Редагування історії освіти та роботи

3.3. Створення резюме

Застосунок надає можливість експорту даних профілю у вигляді документа, що може використовуватися в якості резюме.

Використовуючи наявні елементи інтерфейсу (рисунок 3.12), користувач може скоригувати вміст документа. Наразі передбачене опційне включення секцій освіти й досвіду роботи. Так само користувач самостійно визначає, які з його умінь вносити до резюме. Для цього створено альтернативне представлення списку навичок, що згруповані за категоріями й можуть обиратися. Запропоновані налаштування дозволяють створити документ за власними потребами. Виключивши розділи про освіту й роботу, можна згенерувати звіт про опановані компетентності. У виборі умінь, які включаються у звіт, слід керуватися вимогами позиції, для котрої цей звіт генерується.

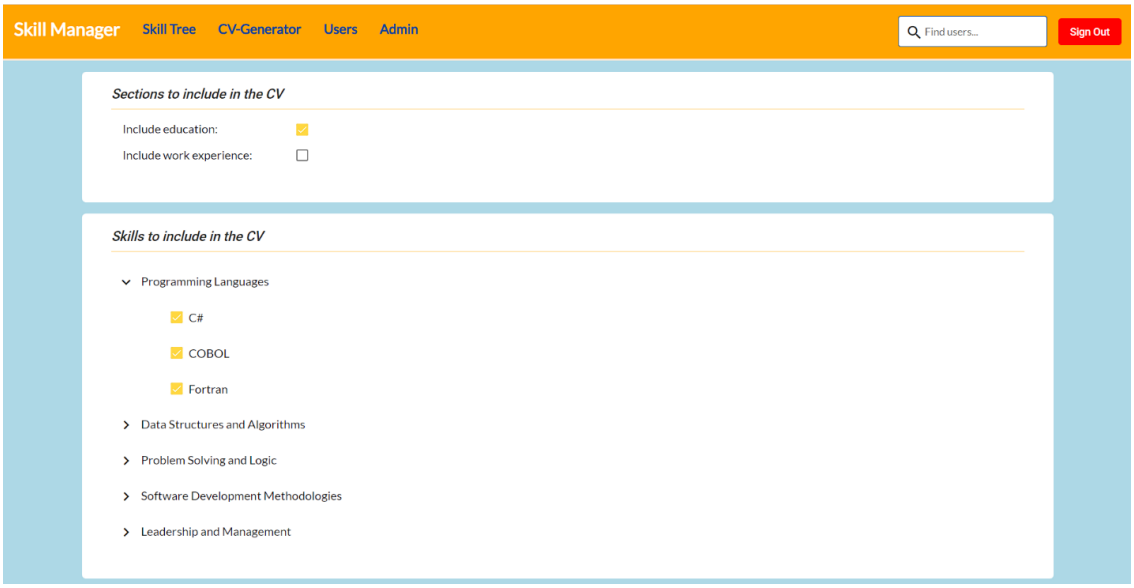


Рисунок 3.12 – Конфігурація резюме

Зафіксувавши бажану конфігурацію, користувачу лишається обрати один із наявних шаблонів документа (рисунок 3.13) й натиснути кнопку «Generate CV». У результаті, генерується документ, доступний користувачу для завантаження. Приклад документа наведено в додатку Д.

Рисунок 3.13 – Вибір шаблону резюме

Адміністратор системи може додавати нові шаблони й видаляти наявні. Шаблон завантажується у вигляді текстового представлення документа XSLT. Після додавання шаблон автоматично стає доступним для генерації.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ СИСТЕМИ

В результаті роботи розроблено систему, що складається із трьох сервісів, кожен із яких реалізує деяку частину функціонала застосунку. Графічний інтерфейс користувача забезпечує SPA-клієнт, що обмінюється повідомленнями із сервісами за посередництва API Gateway. На рис. 4.1 зображено діаграму загальної організації системи.

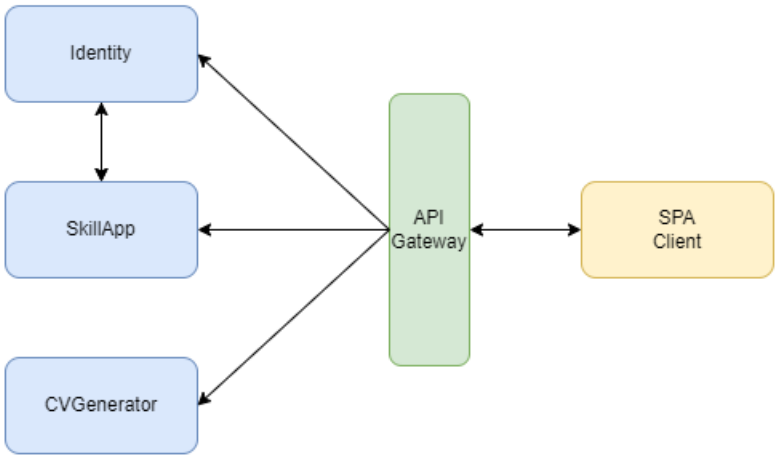


Рисунок 4.1 – Структура розробленої системи

Складові компоненти системи мають наступне призначення:

- SkillApp – додавання й перегляд навичок, оцінка умінь користувачів, робота із інформативними даними користувачів;
- Identity – автентифікація та реєстрація користувачів системи, реалізація token-based автентифікації з refresh токеном;
- CVGenerator – створення й експорт резюме користувачів згідно з надісланими даними відповідно до обраного шаблону.

Наведемо огляд реалізації кожного з компонентів системи, а також застосованих технологій та підходів проєктування.

4.1. Огляд сервісу SkillApp

Мікросервіс SkillApp реалізує головну частину функціонала системи, що виражається в роботі із навичками користувачів. Додаток Б демонструє знімок сторінки OpenAPI специфікації сервісу. Структура БД зображена в Додатку В.

Упроваджені паттерни та підходи проєктування: архітектура портів і адаптерів, DDD, CQRS, шаблони «Робоча одиниця», «Специфікація», «Репозиторій», «Ланцюжок обов'язків», «Медіатор».

Структура сервісу та технології. SkillApp складається із трьох бібліотек класів .NET і вебпроєкту, що використовує ASP.NET WebApi для створення REST API. Дані сервіс зберігає в БД MS SQL Server, застосовуючи ORM-технологію Entity Framework. Бібліотека MediatR використана для забезпечення вільного зв'язку між контролерами та реалізацією логіки. Для відображення сутностей різних шарів проєкту використано AutoMapper. Логування забезпечене бібліотекою Serilog.

На рис. 4.2 зображена діаграма модулів, які складають сервіс SkillApp.

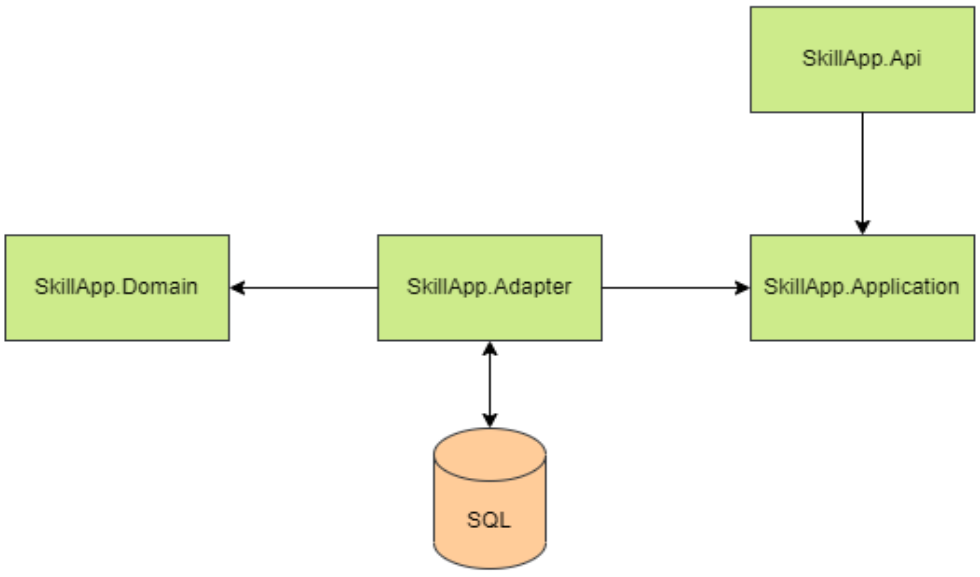


Рисунок 4.2 – Структура сервісу SkillApp

Сервіс реалізує так звану «шестикутну архітектуру», що відмежовує бізнес-логіку від способу зберігання даних. В рамках її застосування розроблено наступні модулі мікросервісу, короткий опис яких наведено в таблиці 4.1.

Таблиця 4.1 – Модулі мікросервісу SkillApp

Назва	**Тип**	**Опис**
SkillApp.Domain	Бібліотека класів	Визначає доменні сутності. Модуль є повністю самодостатнім і не залежить від жодного іншого модулю
SkillApp.Application	Бібліотека класів	Реалізує бізнес-логіку сервісу. Він залежить лише від домену і надає вимоги до потенційної реалізації сховища даних у вигляді інтерфейсів.
SkillApp.Adapter	Бібліотека класів	Містить конфігурацію таблиць для збереження доменних сутностей в реляційній БД. Проєкт виступає в ролі адаптера між доменом та бізнес-логікою. Він може бути безперешкодно замінений іншою реалізацією при потребі зміни способу збереження даних.
SkillApp.Api	REST API	Зовнішній інтерфейс мікросервісу у вигляді REST API. Комунікує із SkillApp.Application, надсилаючи запити та команди через медіатор.
SQL Server	База даних	Реляційне сховище даних, структура таблиць якого сконфігурована з використанням підходу Code First.

Реалізація моделі домену. Важливим питанням при впровадженні підходу DDD є проблема місця логіки запитів, сортування й пагінації. Відповіддю на це питання може стати шаблон «Специфікація». «Паттерн «Специфікація» дозволяє інкапсулювати частину знань про предметну область в одну одиницю (специфікацію) і повторно використовувати її в різних частинах кодової бази» – так цей шаблон описано у статті [17]. Відповідно до паттерну, в проєкті SkillApp.Dimain реалізовано інтерфейс ISpecification (рисунок 4.3). Він дозволяє на рівні домену іменувати можливі запити до сховища даних.

```
public interface ISpecification<T>
{
    Expression<Func<T, bool>> Criteria { get; }

    List<Expression<Func<T, object>>> Includes { get; }

    List<string> IncludeStrings { get; }
}
```

Рисунок 4.3 – Інтерфейс ISpecification

Приклад специфікації запиту для отримання об’єктів сутності Appraisal (оцінка рівня володіння навичкою користувачем) наведемо на рис. 4.4.

```
public class AppraisalSpecification : Specification<Appraisal>
{
    public AppraisalSpecification(string appraiseeId, int skillId)
        : base(c => c.AppraiseeId == appraiseeId && c.SkillId == skillId)
    {
    }

    public AppraisalSpecification(string appraiserId, string appraiseeId)
        : base(c => c.AppraiserId == appraiserId && c.AppraiseeId == appraiseeId)
    {
        this.AddInclude(c => c.Skill);
        this.AddInclude("Skill.Parent");
    }
}
```

Рисунок 4.4 – Специфікація сутності Appraisal

Реалізація логіки. Логіка сервісу реалізована з застосуванням паттерну CQRS, який полягає в логічному розділенні операцій читання й редагування даних системою. При спільній реалізації CQRS та паттерну «Репозиторій» були створені окремі репозиторії для читання й редагування даних, методи яких використовують згадані раніше класи специфікацій для запитування даних.

Для збереження внесених змін різних таблиць у рамках єдиної транзакції використано паттерн «Робоча одиниця». Його застосування дозволяє спершу проводити зміни усіх необхідних даних в оперативній пам'яті й лише потім контрольовано й централізовано зберігати їх у постійне сховище.

Використання MediatR. Іншою складовою реалізації CQRS стало використання бібліотеки MediatR. Вона дозволяє явно визначити операції читання даних – запити (Queries) та їх зміни – команди (Commands). Для кожної команди необхідно реалізувати два класи: безпосередньо команда та її обробник. Для запитів також додається третій клас, що містить відповідь на запит. На рис. 4.5. наведемо список реалізованих команд та запитів у мікросервісі SkillApp.

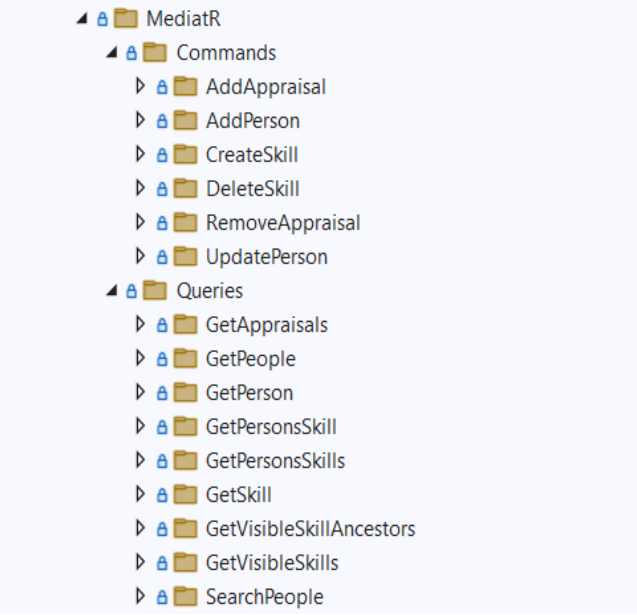


Рисунок 4.5 – Команди й запити сервісу SkillApp

Утім, окрім свого безпосереднього завдання, бібліотека MediatR містить низку додаткових можливостей, пов'язаних із побудовою пайплайну обробки запитів і команд. В проєкті вони використані для вирішення завдань відображення сутностей (mapping), авторизації й отримання даних про автентифікованого користувача.

Реалізація прав користувача. Авторизація в сервісі SkillApp реалізована на рівні бізнес-логіки із застосуванням паттерну «Ланцюжок обов'язків». Вона базується на перевірці визначених правил доступу до функцій сервісу. Абстрактний загальний клас AuthorizationRuleBase (Додаток Г) містить декларацію методу EvaluateThis, у якому дочірні класи мають визначити перевірку певного правила авторизації. Правила можуть об'єднуватися в ланцюг, використовуючи посилання на наступне й попереднє правило. Метод Evaluate виконує перевірку попереднього правила і виклик методу EvaluateThis для перевірки власного правила.

Класи запитів та команд, для яких необхідно виконати авторизацію, реалізують інтерфейс IAuthorizedRequest (рисунок 4.6). Таким чином вони містять властивість для зберігання списку правил.

```
public interface IAuthorizedRequest<TRequest> where TRequest : IRecognizable
{
    AuthorizationRuleBase<TRequest>[] AuthorizationRules { get; }
}
```

Рисунок 4.6 – Інтерфейс запиту, що потребує авторизації

У якості прикладу реалізації автентифікаційного правила наведемо клас UserIsSkillCreator, який перевіряє, чи є автентифікований користувач творцем даної навички (рисунок 4.7). Виконання перевірки правил авторизації відбувається при обробці запиту в пайплайні MediatR.

```
public class UserIsSkillCreator<TRequest> : AuthorizationRuleBase<TRequest> where
                                    TRequest : ISkillRequest, IRecognizable
{
    private readonly ISkillService _skillService;

    public UserIsSkillCreator(IServiceProvider serviceProvider) {…}

    protected override async Task<bool> EvaluateThis(TRequest request)
    {
        _errorMessage += $" User {request.User.UserId} is not the creator of the
                                                    skill {request.SkillId}.";

        return (await _skillService.GetSkillAsync(request.SkillId))?.CreatorId ==
                                                            request.User.UserId;
    }
}
```

Рисунок 4.7 – Правило перевірки авторства навички

Наведемо реалізацію команди видалення навички (рисунок 4.8). Реалізувавши необхідні інтерфейси, до цієї команди застосовна перевірка правил авторизації. Властивість AuthorizationRules містить правила перевірки, чи є користувач адміністратором або творцем уміння, що видаляється.

```
public record DeleteSkillCommand(int SkillId, IServiceProvider ServiceProvider)
    : IRecognizable, IRequest<bool>, ISkillRequest,
      IAuthorizedRequest<DeleteSkillCommand>, IRequestDI
{
    public UserClaims User { get; set; } = null!;

    public AuthorizationRuleBase<DeleteSkillCommand>[] AuthorizationRules => new[]
    {
        new UserIsAdminRule<DeleteSkillCommand>().Or(new
                    UserIsSkillCreator<DeleteSkillCommand>(ServiceProvider)),
    };
}
```

Рисунок 4.8 – Команда видалення навички

Винесення логіки на рівень Application, централізоване відображення моделей та перевірка авторизаційних правил дозволили створити «тонкі» контролери на рівні API, які містять мінімум коду. Типовий приклад методів контролерів наведено на рис. 4.9. Вони полягають у виклику запиту або команди

через медіатор, якому передаються параметри включно з бажаним типом отримуваної моделі.

```
[HttpGet("{skillId}")]
public async Task<IActionResult> GetSkillAsync([FromRoute] int skillId)
{
    var skill = await _mediator.Send(new GetSkillQuery(skillId,
                typeof(GetSkillViewModel), HttpContext.RequestServices));

    return skill.MappedResult is null ? NotFound() : Ok(skill.MappedResult);
}

[HttpPost]
public async Task<IActionResult> CreateSkillAsync(
            [FromBody] CreateSkillViewModel skillViewModel)
{
    return Ok((await _mediator.Send(
                new CreateSkillCommand(skillViewModel))));
}
```

Рисунок 4.9. Методи GET і POST контролеру навичок

4.2. Огляд сервісу Identity

Мікросервіс Identity додано для відокремлення процесу автентифікації та реєстрації користувачів. Він реалізує token-based авторизацію із refresh-токеном.

Упроваджені паттерни та підходи проєктування: архітектура портів і адаптерів, паттерн «Репозиторій».

У межах сервісу реалізовано два контролери. Перший відповідає за реєстрацію, вхід до акаунту й оновлення токена. Другий призначений для роботи з ролями. Знімок специфікації реалізованого API наведена на рис. 4.10.

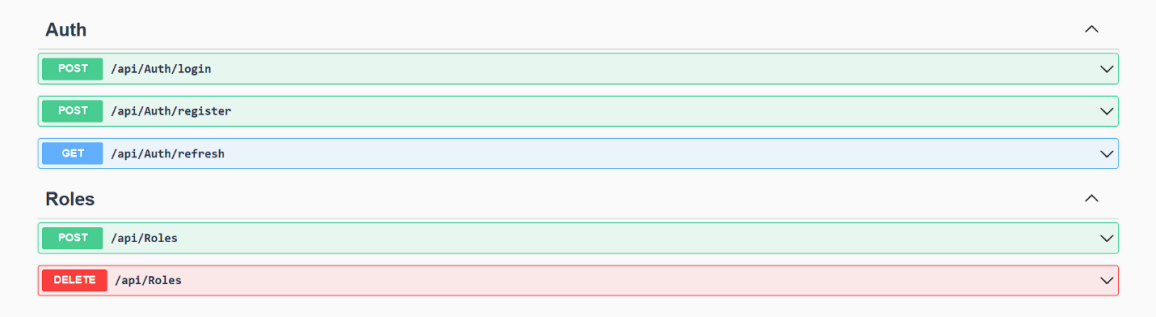


Рисунок 4.10 – API сервісу Identity

БД даного мікросервісу (рисунок 4.11) має просту структуру. Вона відведена виключно для зберігання автентифікаційних даних і ролей користувачів.

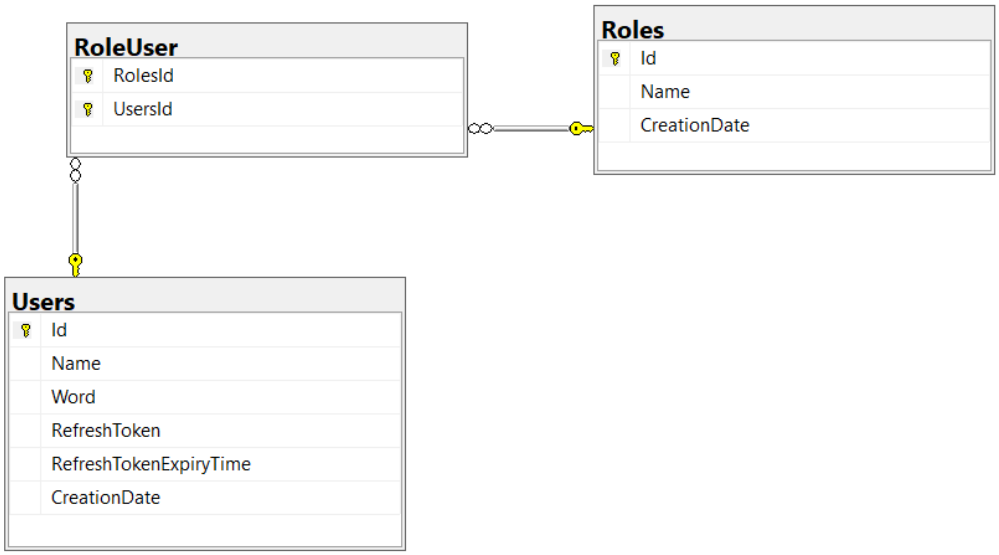


Рисунок 4.11 – Схема БД сервісу Identity

Структура сервісу та технології. Як і розглянутий раніше мікросервіс SkillApp, Identity складається із трьох бібліотек класів і веб-API, що організовані у вигляді архітектури портів і адаптерів. Розроблений із використанням технологій .NET, ASP.NET і Entity Framework. Для синхронізації із компонентом SkillApp при реєстрації користувача використано gRPC. Короткий опис кожної складової частини сервісу наведемо в таблиці 4.2.

Таблиця 4.2 – Модулі мікросервісу Identity

Назва	Тип	Опис
Auth.Domain	Бібліотека класів	Містить опис доменних сутностей користувача та ролі.
Auth.Application	Бібліотека класів	Реалізує логіку додавання й видалення користувачів, управління ролями та генерації access- і refresh-токенів.
Auth.Adapter	Бібліотека класів	Відповідає за збереження даних у реляційній БД. Містить конфігурацію таблиць і реалізацію репозиторіїв.
Auth.Api	REST API	Надає вебінтерфейс для операцій з автентифікації та реєстрації, а також управління ролями.
SQL Server	База даних	Створене за допомогою підходу Code First реляційне сховище даних.

Опис процесу автентифікації. Реалізована автентифікація на основі токенів – це метод автентифікації користувача, що передбачає використання токена замість імені користувача та пароля при обробці запитів. Після успішної автентифікації користувач отримує токен, який потім включається в наступні запити для підтвердження особи. Цей токен зберігається в локальному сховищі бравзера.

Токен доступу, що відкрито зберігається на боці користувача є незахищеним і може бути використаним зловмисниками. Тому таким токенам надається короткий термін дії, після завершення якого користувач має знову підтвердити свою особу. Утім, повторне введення імені та паролю заважало б зручному користуванню вебсайтом, тому в проєкті використано інший підхід. Він полягає у

використанні другого токену, призначення для оновлення доступу (refresh token), що видається при першій автентифікації користувача. Даний токен має довший термін дії і використовується для автоматичного оновлення токена доступу. Він зберігається в базі даних на боці сервера й передається клієнтом у захищених HTTP-only cookies.

Наведемо короткий огляд процесу автентифікації на основі токенів за допомогою токенів оновлення:

1. Користувач входить в систему і надає свій логін і пароль.
2. Сервер генерує токен доступу з необхідними даними і токен оновлення.
3. Токен доступу зберігається в локальному сховищі браузера, а токен оновлення записується в базу даних.
4. При обробці запиту сервер щоразу перевіряє коректність токена доступу і наявність токена оновлення.
5. При виникненні помилки авторизації клієнт надсилає запит на оновлення токена доступу.
6. Сервер перевіряє відповідність надісланого клієнтом токена оновлення до значення в БД.
7. Якщо перевірка успішна, клієнт отримує новий токен доступу. Інакше користувач має повторити ручну автентифікацію.

Такий підхід забезпечує кращий користувацький досвід, дозволяючи користувачеві залишатися в системі протягом тривалого часу без необхідності повторно вводити свої облікові дані. Він також забезпечує підвищену безпеку, оскільки токени доступу мають короткий термін дії, що зменшує вікно можливостей для зловмисника отримати доступ до облікового запису користувача.

4.3. Огляд сервісу CVGenerator

Даний мікросервіс виконує задачу генерування резюме користувача на основі надісланих даних. Реалізовано кінцеві точки для генерування документа, отримання й додавання шаблонів (рисунок 4.12). Для реалізації створення резюме застосовано паттерн «Шаблонний метод».

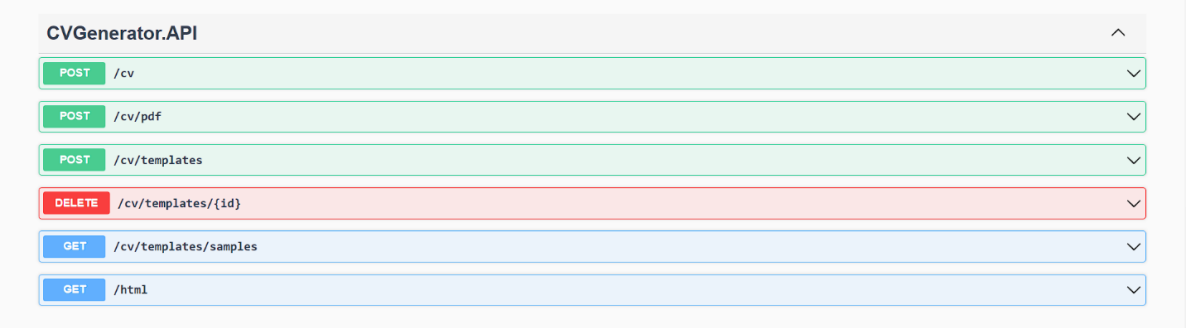


Рисунок 4.12 – API сервісу CVGenerator

Сховище даних NoSQL представлено єдиним типом документу для збереження XSL-шаблону резюме (рисунок 4.13). Даний сервіс не зберігає особисті дані користувачів, тому щоразу генерує документ із надісланими в запиті значеннями.

```
_id: ObjectId('643ac7144b1c79cb07e4f5eb')
xsl: "<?xml version="1.0" encoding="UTF-8"?><xsl:stylesheet version="1.0" xm…"

_id: ObjectId('645b95f16913674a6217321a')
xsl: "<?xml version="1.0" encoding="UTF-8"?><xsl:stylesheet version="1.0" xm…"

_id: ObjectId('6463cfc0945ecf5b61a9c360')
xsl: "<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="1.0" x…"
```

Рисунок 4.13 – Зберігання шаблонів резюме

Структура сервісу та технології. CVGenerator (рисунок 4.14) складається з єдиної бібліотеки класів і проєкту веб-API. API побудовано з використанням Minimal API – способу створення інтерфейсу без створення контролерів, запропонованого в ASP.NET 6. Базовою технологією є .NET, а дані зберігаються в документно-орієнтованій базі даних MongoDB. Для створення HTML-резюме користувачів застосовано технологію XSLT.

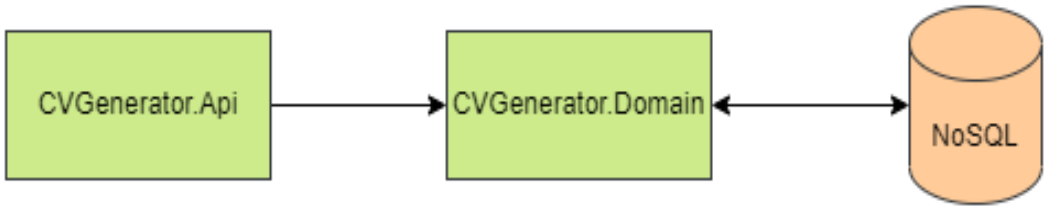


Рисунок 4.14 – Структура сервісу CVGenerator

Призначення компонентів мікросервісу опишемо в таблиці 4.3.

Таблиця 4.3 – Модулі мікросервісу CVGenerator

Назва	Тип	Опис
CVGenerator.Domain	Бібліотека класів	Містить визначення сутностей та реалізацію збереження даних.
CVGenerator.Api	REST API	Представляє зовнішній інтерфейс сервісу, виконаний з підходом Minimal Api. Також реалізує логіку генерації резюме.
MongoDB	База даних	Документно-орієнтоване сховище даних, використане для збереження завантажених користувачем шаблонів резюме.

Огляд процесу генерування резюме. Процес створення документа, що може бути використаний користувачем у якості резюме, проходить у два етапи:

1. Підготовка даних у форматі XML;
2. Генерація HTML-документа.

Необхідність виконання першого етапу викликана використанням технології XSL Template для створення документа. В процесі створення документа задіяні бібліотеки System.Xml, System.Xml.Linq і System.Xml.Xsl. Паттерн проєктування «Шаблонний метод» був застосований при декларації та реалізації кроків підготовки даних резюме у форматі XML. На рис. 3.15 наведено відповідний метод, який передбачає виклик функцій підготовки різних складових резюме, реалізація яких може різнитися.

```
public XDocument PrepareDocData(CVModel model)
{
    var xdocument = new XDocument(
        new XElement("cv",
            PreparePerson(model.Person),
            new XElement("introduction", model.Introduction),
            new XElement("education", PrepareEducation(model.Education)),
            new XElement("experience", PrepareExperience(model.Experience)),
            new XElement("skills", PrepareSkills(model.Skills))));
    return xdocument;
}
```

Рисунок 3.15 – Покрокова підготовка даних для резюме

У додатку Е наведені сформовані методом XML-дані для генерації документа. Сам документ, зібраний за допомогою цих даних, можна побачити в додатку Д.

4.4. Компонент Gateway

Метою використання компонента Gateway є спрощення комунікації клієнта із кількома серверними API. Цей модуль розробленої системи надає єдиний інтерфейс для роботи із трьома сервісами. Завдяки його застосуванню клієнтська програма не залежить від організації компонентної архітектури серверної частини. В майбутньому додавання нового функціоналу не спричинятиме необхідності зміни конфігурації клієнтських програм незалежно від того, чи реалізується цей функціонал наявними сервісами або новоствореними. В будь-якому випадку доступ до них надаватиметься через Gateway.

Для створення уніфікованого інтерфейсу використано бібліотеку Ocelot. У створений спеціально вебпроєкт на ASP.NET додано файл конфігурації ocelot.json. Він містить співставлення шляхів запитів до Gateway із шляхами до відповідних мікросервісів. Приклад такої конфігурації для сервісів, розгорнутих локально, зображено на рис. 4.16. Згідно з нею, GET і DELETE HTTPS-запити до /gateway/skills переадресовуватимуться на /api/skills, відповідно до конфігурації.

```
{
 "DownstreamPathTemplate": "/api/skills/{skillId}",
 "DownstreamScheme": "https",
 "DownstreamHostAndPorts": [
     {
         "Host": "localhost",
         "Port": 7001
     }
 ],
 "UpstreamPathTemplate": "/gateway/skills/{skillId}",
 "UpstreamHttpMethod": [ "GET", "DELETE" ]
}
```

Рисунок 4.16 – Переадресування запитів до навичок

4.5. Компонент SkillApp-Client

Графічний інтерфейс системи забезпечується SPA-застосунком, який спирається на можливості бравзера для комунікації користувача з системою.

Структура компонента та технології. Застосунок створено за допомогою вебфреймворку Angular, що підтримує розробку SPA-застосунків. Інтерактивні елементи інтерфейсу додані з використанням бібліотеки компонентів Angular Material, стилізованих під власні потреби стилями CSS. Асинхронне виконання запитів забезпечене можливостями бібліотеки RxJS і мови TypeScript.

Складовими частинами Angular-проєкту є компоненти, кожен з яких реалізує деякий елемент або сторінку [18]. Кожен із компонентів складається з трьох елементів:

3. TypeScript-класу, що реалізує логіку компонента, зберігає необхідні змінні.
4. Темплейта, який є HTML-сторінкою та може містити директиви й відображення змінних TS-класу. Директиви слугують для зміни структури та поведінки елементів темплейта.
5. CSS-селектору, за допомогою якого компонент може використовувати інші компоненти в якості дочірнього.
6. Опціонально, файлу стилів CSS, які визначають зовнішній вигляд компонента.

Реалізація застосунку. Angular дозволяє створювати компоненти, які можуть вбудовуватися в структуру інших сторінок. Гнучкість перевикористання компонентів забезпечена можливістю додавання вхідних і вихідних властивостей компонента. Дочірній компонент можна сконфігурувати через батьківський, передавши потрібні значення наявних вхідних властивостей.

Наочним прикладом багаторазового застосування є компонент level-bar, призначений для зображення й редагування оцінки рівня володіння умінням користувача. Він застосовується в декількох представленнях. У користувацькому профілі індикатор набуває найбільш повного вигляду, показуючи також оцінку уміння іншими користувачами (рисунок 3.9). На інших сторінках доступний лише перегляд і встановлення власної оцінки. Інший приклад універсальних компонентів: поля для введення тексту, кнопки інтерфейсу. Їх перевикористання полегшує дотримання єдиного стилю в застосунку.

Комунікація графічної частини клієнту з сервером забезпечена процесом, що складається з трьох основних елементів: HTML-темплейта компоненту, його TS-класа і сервісу з HTTP-клієнтом.

Винесення логіки взаємодії з API в окремі класи сервіси сприяє уникненню дублювання коду, адже ці методи зазвичай викликаються в декількох компонентах застосунку. Рисунок 4.17 демонструє приклад використання HTTP-клієнта для надсилання запитів створення навички й отримання вибірки дочірніх умінь.

```
public getChildrenFiltered(skillFilter: SkillFilterModel): Observable<Skill[]> {
    let params = new HttpParams()
      .set("parentId", skillFilter.parentId ?? "")
      .set("appraiseeId", skillFilter.appraiseeId ?? "")
      .set("appraiserId", skillFilter.appraiserId ?? "");
    return this.http.get<Skill[]>(${this.apiUrl}/${this.url},
                  { withCredentials: true, params: params });
  }

public createSkill(skill: CreateSkillModel): Observable<boolean> {
    return this.http.post<boolean>(
             ${this.apiUrl}/${this.url}, skill, { withCredentials: true });
  }
```

Рисунок 4.17 – Застосування HTTP-клієнта

Наявність вбудованого HTTP-клієнта є однією з переваг Angular над іншими вебфреймворками. Він підтримує різні методи запитів, такі як GET, POST, PUT,

DELETE тощо, а також дозволяє додавати заголовки й параметри запиту та його тіло. До того ж HttpClient надає можливості для читання статусів, вмісту відповідей і їх заголовків. JSON відповіді розбираються автоматично, доступне також налаштування для інших форматів. Результат виконання запиту HTTP-клієнт повертає у вигляді об’єкта Observable – ключового класу бібліотеки RxJS.

RxJS є потужним засобом, що надає широкий набір функцій для роботи з асинхронними подіями та даними, джерелом яких можуть бути й HTTP-запити. Бібліотека реалізує механізм підписки на об’єкт Observable, який може продукувати події з плином часу. Підписані обробники подій реагують на них, допоки підписка не зупинена або об’єкт не знищений. Фактично, описаний процес є реалізацією шаблону «Спостерігач» – поведінкового паттерну GoF. Окрім простої підписки на події, RxJS дозволяє вибудовувати послідовну обробку подій. «Маніпулюючи даними в процесі, ви можете адаптувати вихід виробника відповідно до очікувань споживача. Це сприяє розподілу відповідальності між двома суб'єктами, і це велика перевага для модульності вашого коду», - наголошено в книзі [19].

Кожен компонент Angular так чи інакше користується механізмом підписок. Кількість підписок із часом роботи застосунку може зростати, впливаючи на його продуктивність і завантаженість пам’яті. Тому важливо контролювати час існування підписки та закривати її, коли вона більше не потрібна. Очищення підписок і слухачів подій гарантує, що застосунок виконує лише необхідні функції, покращуючи його ефективність. Процес відписування в проєкті було автоматизовано з використанням засобів RxJS. Приклад цього наведено на рис. 4.18.

```
private destroy$: ReplaySubject<void> = new ReplaySubject<void>();

public getChildren(): void {
  this.skillService
    .getChildren(this.skill.id)
    .pipe(takeUntil(this.destroy$))
    .subscribe(res => {
      this.skillChildren = res;
    })
}

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

Рисунок 4.18 – Автоматичне очищення пам’яті

В процес виконання запиту за допомогою оператора pipe додано обробник takeUntil, який продукує значення допоки не отримає сигнал від наданого йому об’єкта Observable. Таким об’єктом в прикладі виступає змінна destroy$. Використовуючи хук-метод життєвого циклу компонента ngOnDestroy [20], виконується надсилання сигналу через destroy$ для автоматичного припинення дії всіх підписок цього компонента. Цей підхід суттєво спрощує контроль за пам’яттю програми.

ВИСНОВКИ

У результаті було розроблено застосунок для ведення обліку професійних навичок користувача. Він дозволяє створювати уміння, додавати їх до власного публічного профіля, оцінювати знання інших користувачів. Для розробки були досліджені й використані підходи проєктування мікросервісних систем.

При плануванні організації системи було прийняте рішення створити окремі сервіси для реалізації функцій системи, що не є тісно пов’язаними між собою. Це вилилося у розробку трьох REST API:

- SkillApp, який пропонує засоби для роботи з навичками та публічними даними користувачів;
- Identity – сервіс, що забезпечує token-based автентифікацію, реєстрацію користувачів і зберігання даних для входу;
- CVGenerator, призначений для генерування резюме користувачів на базі збережених шаблонів.

Реалізація згаданих вище компонентів системи передбачила розробку окремої БД для кожного сервісу для зберігання призначених йому даних. Сервіси SkillApp і Identity покладаються на реляційне сховище даних SQL Server, коли CVGenerator використовує NoSQL MongoDB.

Сервіс SkillApp, як основний носій бізнес-логіки системи, створений розширюваним і підтримуваним. Його розробка базувалась на створенні моделі домену предметної області із застосуванням шаблону «Специфікація». На рівні бізнес-логіки забезпечено розділення операцій читання й запису, згідно CQRS. Слідуючи принципам архітектури портів і адаптерів, далі був створений адаптер для сховища даних та презентаційний рівень у вигляді API.

Об’єднаний доступ до розрізнених інтерфейсів системи забезпечено API Gateway. Наразі він використовується SPA-клієнтом для взаємодії із серверною частиною системи.

Розроблена система має перспективи розвитку свого функціоналу. Серед реальних майбутніх розширень такі:

- можливість створення навчальних планів і довільне групування навичок;
- можливість комунікації користувачів всередині застосунку, підтримка спільнот за інтересами;
- інтеграція з сервісами пошуку роботи;
- інтеграція з сервісами онлайн-навчання для підбору курсів і проходження сертифікацій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Djinni. Аналітика Джина [Електронний ресурс] – Режим доступу до ресурсу: https://djinni.co/analytics/supply-demand.

2. Design Patterns: Elements of Reusable Object-Oriented Software / E.Gamma, R. Helm, R. Johnson, J. Vlissides., 1994. – 416 с. – (1st Edition).

3. Skills Base [Електронний ресурс] – Режим доступу до ресурсу: https://www.skills-base.com/.

4. LinkedIn [Електронний ресурс] – Режим доступу до ресурсу: https://www.linkedin.com/.

5. Zety [Електронний ресурс] – Режим доступу до ресурсу: https://zety.com/.

6. Whitesell S. Pro Microservices in .NET 6: With Examples Using ASP.NET Core 6, MassTransit, and Kubernetes / S. Whitesell, R. Richardson, M. D. Groves.. – 320 с. – (1st ed. Edition).

7. Agile Teams [Електронний ресурс] – Режим доступу до ресурсу: https://scaledagileframework.com/agile-teams.

8. Burns B. Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services / Brendan Burns., 2018. – 162 с. – (1st Edition).

9. Horizontal Scaling vs Vertical Scaling [Електронний ресурс] – Режим доступу до ресурсу: https://openmetal.io/docs/edu/openstack/horizontal-scaling-vs-vertical-scaling/.

10. The pros and cons of the CQRS architecture pattern [Електронний ресурс] – Режим доступу до ресурсу: https://www.redhat.com/architect/pros-and-cons-cqrs.

11. Henrique D. S. The Command and Query Responsibility Segregation (CQRS) Pattern [Електронний ресурс] / Domareski Siebert Henrique – Режим

доступу до ресурсу: https://henriquesd.medium.com/the-command-and-query-responsibility-segregation-cqrs-pattern-16cb7704c809.

12. Palermo J. The Onion Architecture : part 1 [Електронний ресурс] / Jeffrey Palermo – Режим доступу до ресурсу: https://jeffreypalermo.com/2008/07/the-onion-architecture-part-1/.

13. Martin R. C. Design Principles and Design Patterns / Robert Martin.. – 34 с.

14. Halili V. Hexagonal architecture: What is it and why should you use it? [Електронний ресурс] / Vullkan Halili – Режим доступу до ресурсу: https://cardoai.com/what-is-hexagonal-architecture-should-you-use-it.

15. Single-page application vs. multiple-page application [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58.

16. SPA vs. MPA: Pros, Cons & How To Make Final Choice [Електронний ресурс] – Режим доступу до ресурсу: https://www.simicart.com/blog/spa-vs-mpa.

17. Santos R. .NET Core — Using the Specification pattern alongside a generic Repository [Електронний ресурс] / Rodrigo Santos – Режим доступу до ресурсу: https://medium.com/@rudyzio92/net-core-using-the-specification-pattern-alongside-a-generic-repository-318cd4eea4aa.

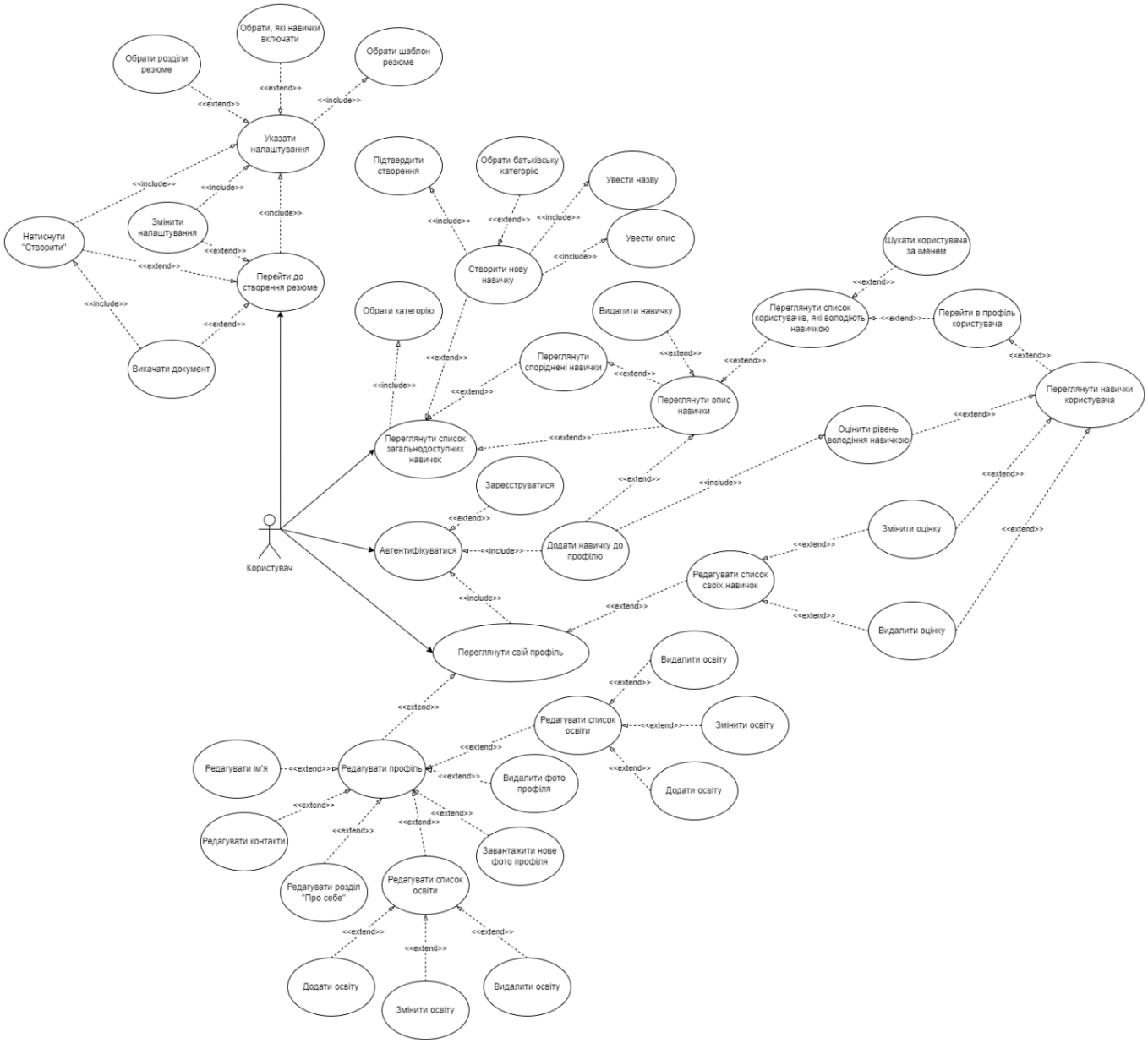
18. Hussain A. Angular from theory to practice / Asim Hussain.. – 692 с. – (Version 1.2.0).

19. Daniels P. P. RxJS in Action / P. P. Daniels, L. Atencio., 2017. – 352 с.

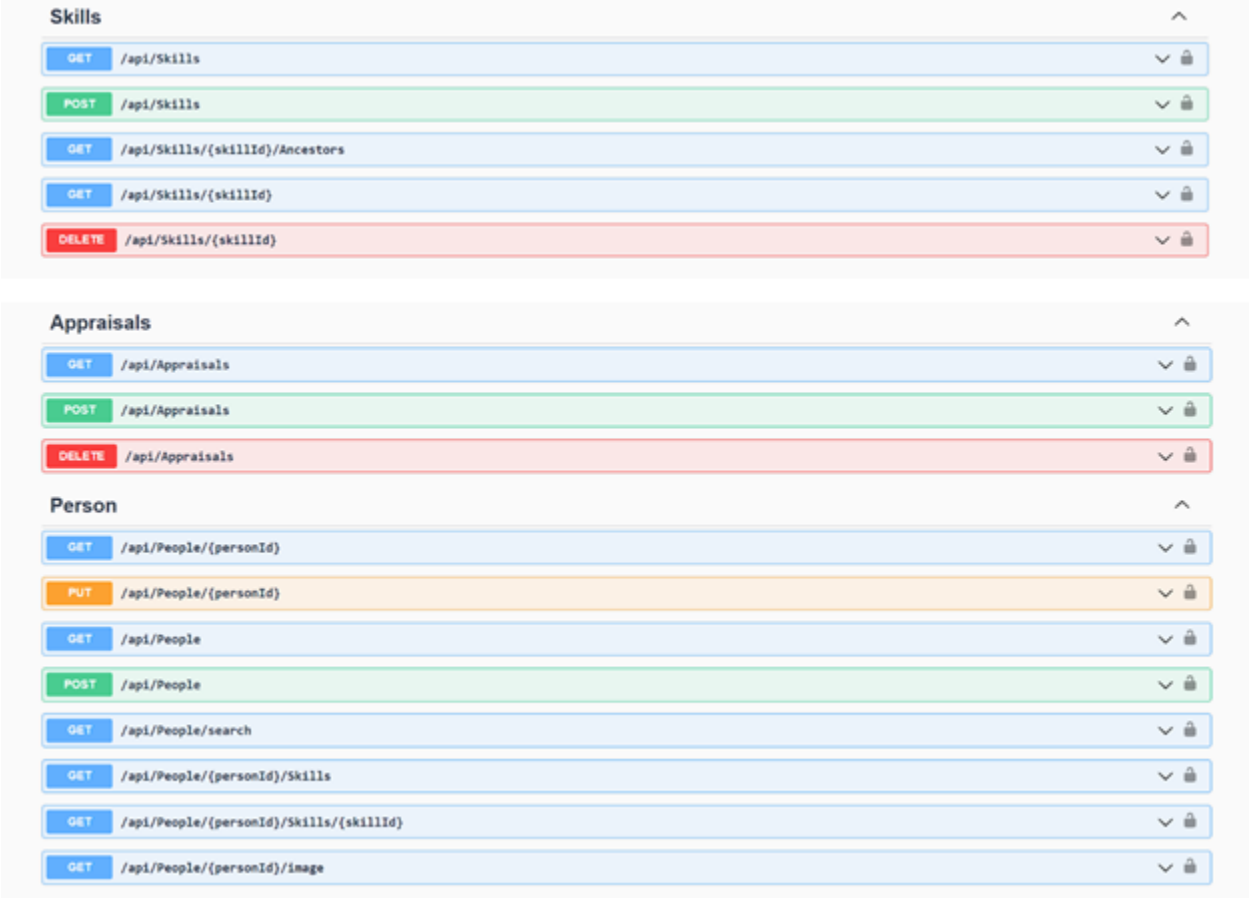
20. ng-book: The Complete Guide to Angular / N.Murray, F. Coury, A. Lerner, C. Taborda., 2018. – 626 с. – (5th Edition).

ДОДАТКИ

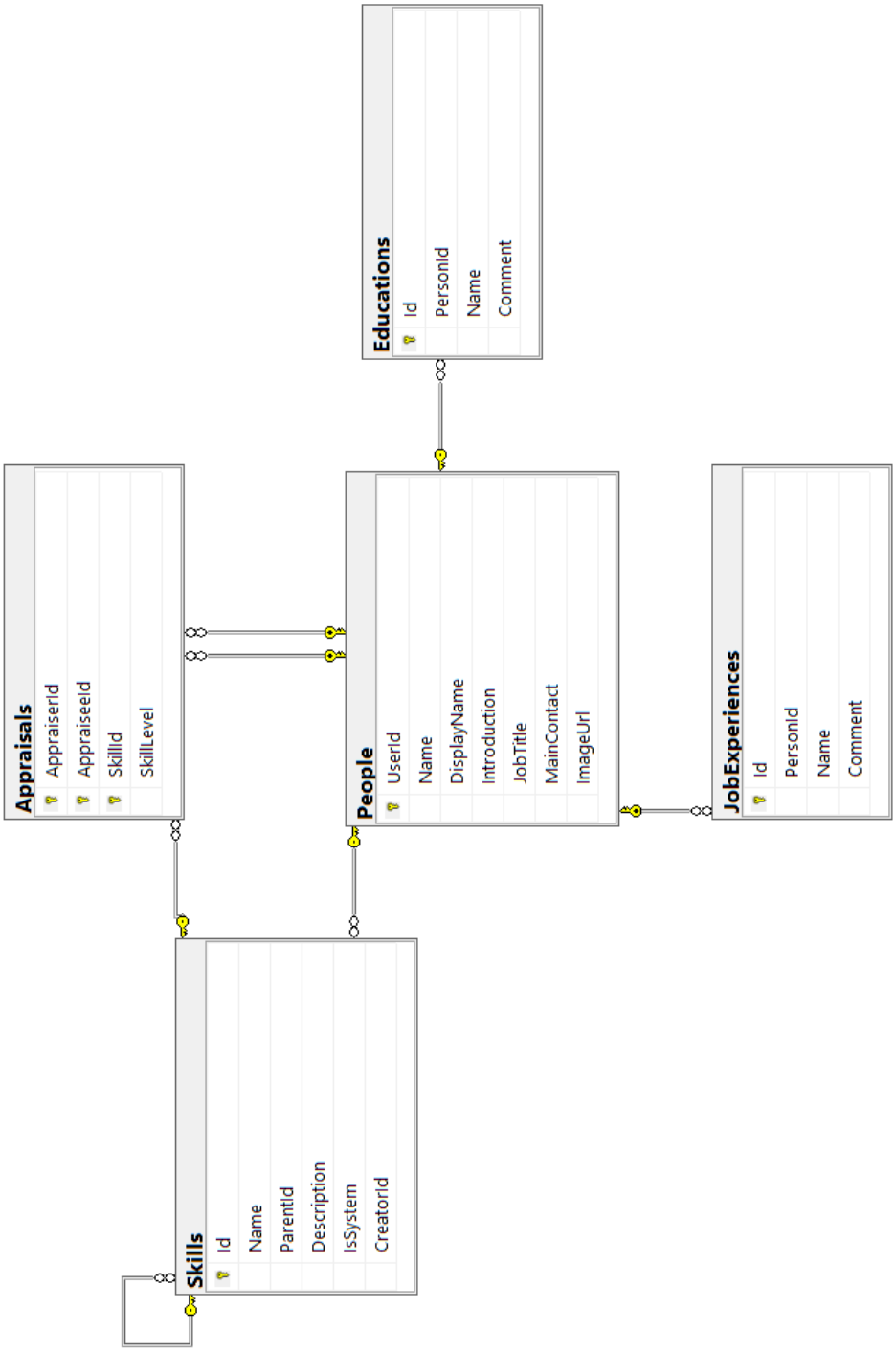
Додаток А. Діаграма прецедентів



Додаток Б. API сервісу SkillApp



Додаток В. Діаграма БД сервісу SkillApp



Додаток Г. Реалізація правила доступу

```
      public abstract class AuthorizationRuleBase<TRequest> where TRequest :
IRecognizable
      {
          public AuthorizationRuleBase<TRequest> Next { get; private set; } = null!;

          public AuthorizationRuleBase<TRequest> Prev { get; private set; } = null!;

          protected string _errorMessage = string.Empty;

          public async Task Evaluate(TRequest request)
          {
              try
              {
                  if (Prev is not null)
                  {
                      await Prev.Evaluate(request);
                      return;
                  }
              }
              catch (ForbiddenException ex)
              {
                  _errorMessage += $" {ex.Message}";
              }

              if (!(await EvaluateThis(request)))
              {
                  throw new ForbiddenException(_errorMessage);
              }
          }

          protected abstract Task<bool> EvaluateThis(TRequest request);

          public AuthorizationRuleBase<TRequest> Or(AuthorizationRuleBase<TRequest>
next)
          {
              Next = next;
              Next.Prev = this;
              return Next;
          }
      }
```

Додаток Д. Згенерований документ резюме

Gandalf

Wizard

CONTACT

gandalfwithard@knu.ua

EDUCATION

School of the Elven Sages
Bachelor Degree

Academy of the Dwarven Runesmiths
Master Degree

Tower of the Istari
Doctor Degree

PROFILE INFO

I am Gandalf the Grey, a seasoned and revered wizard from the mystical realm of Middle-earth. With my long, flowing beard and trusty staff in hand, I bring with me a wealth of wisdom and centuries of magical expertise. I have ventured far and wide, encountering mythical creatures and unraveling arcane mysteries. Throughout my storied journey, I have mastered the arts of enchantment, divination, and harnessing the elemental forces of nature. With a steadfast dedication to the pursuit of knowledge and the protection of all that is good, I am prepared to lend my formidable skills and unwavering resolve to any noble quest that lies ahead.

WORK EXPERIENCE

Guardian of the White Council

As a trusted member of the White Council, Gandalf served as a guardian and protector of Middle-earth.

Wanderer and Seeker of Knowledge

Between his grand adventures, Gandalf often embarked on solitary journeys, seeking ancient wisdom and forgotten lore.

Fellowship of the Ring

Gandalf's most notable work experience was his pivotal role as a member of the Fellowship of the Ring.

SKILLS

Programming Languages

- C#
- COBOL
- Fortran

Problem Solving and Logic

- Analytical Thinking
- Algorithm Design

Software Development Methodologies

- Waterfall Model
- Agile (Scrum, Kanban)
- Continuous Integration and Deployment (CI/CD)

Leadership and Management

- Team Building
- Project Management

Додаток Е. Сформовані XML-дані документу


```
<cv>
  <person>
    <imageUrl>https://localhost:7004/gateway/People/1/image</imageUrl>
    <fullname>Gandalf</fullname>
    <jobtitle>Wizard</jobtitle>
    <contacts>
      <main>gandalfwithard@knu.ua</main>
    </contacts>
  </person>
  <introduction> I am Gandalf the Grey, a seasoned and revered wizard from the
mystical realm of Middle-earth. With my long, flowing beard and trusty staff in
hand, I bring with me a wealth of wisdom and centuries of magical expertise. I have
mastered the arts of enchantment, divination, and harnessing the elemental forces of
nature. With a steadfast dedication to the pursuit of knowledge and the protection
of all that is good, I am prepared to lend my formidable skills and unwavering
resolve to any noble quest that lies ahead.</introduction>
  <education>
    <place>
      <name>School of the Elven Sages</name>
      <comment>Bachelor Degree</comment>
    </place>
    <place>
      <name>Academy of the Dwarven Runesmiths</name>
      <comment>Master Degree</comment>
    </place>
    <place>
      <name>Tower of the Istari</name>
      <comment>Doctor Degree</comment>
    </place>
  </education>
  <experience>
    <place>
      <name>Guardian of the White Council</name>
      <comment>As a trusted member of the White Council, Gandalf served as a
guardian and protector of Middle-earth.</comment>
    </place>
    <place>
      <name>Wanderer and Seeker of Knowledge</name>
      <comment>Between his grand adventures, Gandalf often embarked on solitary
journeys, seeking ancient wisdom and forgotten lore.</comment>
    </place>
    <place>
      <name>Fellowship of the Ring</name>
      <comment>Gandalf's most notable work experience was his pivotal role as a
member of the Fellowship of the Ring.</comment>
    </place>
  </experience>
  <skills>
    <category>
      <name>Programming Languages</name>
      <comment />
      <skill>
        <name>C#</name>
```

```
      </skill>
      <skill>ë
        <name>COBOL </name>
      </skill>
      <skill>
        <name>Fortran</name>
      </skill>
    </category>
    <category>
      <name>Problem Solving and Logic</name>
      <comment />
      <skill>
        <name>Analytical Thinking</name>
      </skill>
      <skill>
        <name>Algorithm Design</name>
      </skill>
    </category>
    <category>
      <name>Software Development Methodologies</name>
      <comment />
      <skill>
        <name>Waterfall Model</name>
      </skill>
      <skill>
        <name>Agile (Scrum, Kanban)</name>
      </skill>
      <skill>
        <name>Continuous Integration and Deployment (CI/CD)</name>
      </skill>
    </category>
    <category>
      <name>Leadership and Management</name>
      <comment />
      <skill>
        <name>Team Building</name>
      </skill>
      <skill>
        <name>Project Management</name>
      </skill>
    </category>
  </skills>
</cv>
```