МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ
Навчально-науковий інститут автоматики, кібернетики та обчислювальної техніки
Кафедра комп'ютерних наук та прикладної математики

«До захисту допущений»
Завідувач кафедри комп'ютерних наук та прикладної математики
_____ Турбал Ю.В.
«_____» _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розроблення інформаційно-пошукової системи "Відділ кадрів"»

Виконав: Бойчук Богдан Миколайович _____
(прізвище, ім'я, по батькові) (підпис)

група КН-41

Керівник: професор Бомба А. Я. _____
(науковий ступінь, вчене звання, посада, прізвище та ініціали) (підпис)

Рівне – 2023

Національний університет водного господарства та природокористування

Навчально-науковий інститут автоматики, кібернетики та обчислювальної техніки

Кафедра **комп'ютерних наук та прикладної математики**

Рівень вищої освіти **бакалавр**

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

«ЗАТВЕРДЖУЮ»
Завідувач кафедри Турбал Ю.В.
_____

«____» _____2023 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бойчуку Богдану Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Розроблення інформаційно-пошукової системи "Відділ кадрів"*

керівник роботи *Бомба Андрій Ярославович, професор, доктор технічних наук за спеціальністю "математичне моделювання та обчислювальні методи", кандидат фізико-математичних наук, професор кафедри прикладної математики НУВГП*

затверджені наказом вищого навчального закладу від «*19 » квітня 2023 року*

С №-449

2. Термін здачі студентом закінченої роботи *29.05.2023*
3. Вихідні дані до роботи*: документація замовника*
4. Зміст розрахунково-пояснювальної записки *Розділ 1.*
5. Перелік графічного матеріалу *мультимедійна презентація*
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділи 1, 2, 3, 4	*Бомба А. Я., професор*	10.10.2022	24.12.2022

7. Дата видачі завдання *02.10.2022*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи
1	*Опрацювання літератури та інтернет-джерел на задану тему.*	11.11.2022
2	*Узгодження із замовником.*	15.12.2022
3	*Опрацювання завдання та виявлення ключових аспектів*	13.01.2022
4	*Вибір технологій для реалізації серверної частини*	06.02.2023
5	*Вибір технологій для реалізації клієнтської частини*	22.03.2023
6	*Аналіз та вивчення документації для реалізації*	26.03.2023
7	*Програмна реалізація.*	15.04.2023
8	*Оформлення теоретичної частини*	03.06.2023

Студент _____ (Бойчук Б. М.)

Керівник кваліфікаційної роботи _____ (Бомба А. Я.)

Зміст

РЕФЕРАТ

Мета: Розробити інформаційно-пошукову систему, яка допоможе відділу кадрів у збереженні, обробці та ефективному використанні великого обсягу даних про персонал компанії. Дана система буде містити набір функцій, що спрощують та автоматизують рутинні операції, пов'язані з управлінням кадрами, такі як збір, збереження та оновлення даних співробітників. При розробці системи будуть використані сучасні технології програмування, бази даних та методи пошуку, забезпечуючи швидку та точну обробку запитів користувачів. Основна мета полягає у підвищенні продуктивності та ефективності роботи відділу кадрів, зменшенні часу, затраченого на адміністративні процеси та покращенні якості прийняття рішень в галузі управління персоналом.

Для цього необхідно виконати наступні пункти:

- Вивчення існуючих рішень: Ознайомитися зі схожими системами, що використовуються в інших компаніях або у подібних сферах, зокрема інформаційними системами управління персоналом, пошуковими системами та базами даних.
- Визначення функціональних вимог: Розробити детальний перелік функціональних вимог до інформаційно-пошукової системи, враховуючи потреби відділу кадрів. Вимоги можуть включати збір та збереження даних про співробітників, пошук за різними критеріями, генерацію звітів та інші необхідні функції.
- Проектування бази даних: Розробити структуру бази даних, включаючи таблиці та зв'язки між ними, для збереження та організації даних про співробітників.
- Розробка інтерфейсу користувача: Створити зручний та інтуїтивно зрозумілий інтерфейс, який дозволить користувачам з легкістю взаємодіяти з системою, вводити та отримувати інформацію.
- Розробка пошукової функціональності: Реалізувати механізми пошуку та фільтрації даних, що дозволять відділу кадрів швидко та ефективно знаходити необхідну інформацію.
- Тестування та налагодження: Провести ретельне тестування системи, виявити та виправити помилки та недоліки, що можуть виникнути під час роботи системи.
- Оцінка результатів: Провести оцінку ефективності та користь від використання інформаційно-пошукової системи для відділу кадрів, порівняти її з попередніми методами та системами, і зробити висновки щодо досягнення мети дипломної роботи.

Актуальність: дана тема є доволі важливою на сьогодення, так як в сучасних умовах бізнесу ефективне управління персоналом є вирішальним фактором для досягнення конкурентних переваг. Оптимальне планування робочих місць, відбір, оцінка та розвиток персоналу стають основою успішного функціонування організацій. Потреби в автоматизації та оптимізації процесів стоять гостро. Традиційні методи управління персоналом, такі як ручне ведення паперової документації або використання вузькоспеціалізованих програм, можуть бути часо- та ресурсозатратними. Розробка інформаційно-пошукової системи дозволить автоматизувати багато рутинних процесів, спростити доступ до інформації та покращити продуктивність відділу кадрів.

Результат: виконання цих дій допоможе зрозуміти потреби відділу кадрів, розробити ефективну та зручну систему для управління персоналом і досягти мети дипломної роботи.

Ключові слова: інформаційно-пошукова система, відділ кадрів, ефективність, автоматизація, управління персоналом, вивчення існуючих рішень, функціональні вимоги, проектування бази даних, розробка інтерфейсу користувача, тестування та налагодження, оцінка результатів.

Вступ

Управліннялюдськимиресурсамиєважливимаспектом ефективного функціонування будь-якої організації. Відуспішногопланування роботидо розвитку таутриманняталановитих співробітників,HRвідіграєключовуроль у забезпеченні високої продуктивності та конкурентоспроможності. Однак зростання обсягів даних про працівників таускладнення кадрових процесів ставлять перед HR нові виклики, так як традиційні методи, такі як документообіг та ручний пошук інформації, стають неефективними та затратними з точки зору часу та ресурсів.

У цьому контексті розвиток інформаційно-пошукових систем для відділу кадрів має велике значення. Такі системи автоматизують процеси, забезпечують швидкий і зручний доступ до інформації про співробітників, підвищують ефективність і продуктивність роботи HR-відділів. Метою даної роботи є розробка комплексної інформаційно-пошукової системи, яка використовується відділами кадрів для покращення управління персоналом. Внутрішня частина системи буде реалізована з використанням C#, технології .NET Core та Entity Framework длязабезпечення надійності та масштабованості системи. Фронтенд-частина буде розроблена з використанням технологій Angular, та Node.js для створення зручного та інтуїтивно зрозумілого користувацького інтерфейсу.

Результатом роботи стане інформаційно-пошукова система, яка значно полегшить рутинні HR-процеси, забезпечитьшвидкий доступ до інформації про співробітників та покращить управління персоналом. Розробка такої системи має великий потенціал дляпідвищення продуктивності, ефективності та конкурентоспроможності організації. Окрім цього, існує значна потреба в автоматизації та цифровізації процесів управління персоналом. Традиційні методи, такі як ручне ведення паперової документації або використання вузькоспеціалізованих програм, можуть бути обтяжливими, часо- та ресурсозатратними. Створення інформаційно-пошукової системи для відділу кадрів дозволить перейти до ефективних та автоматизованих процесів управління, що покращить швидкість, точність та надійність обробки інформації. Крім того, розвиток технологій, зокрема мови C#, технології .NET Core та фреймворків Entity Framework для розробки Back-End, а також Angular і Node.js для розробки Front-End, надає потужну та гнучку інструментальну базу для створення сучасних та масштабованих рішень у сфері інформаційних систем. Використання цих технологій дозволить розробити інформаційно-пошукову систему, яка буде ефективно працювати з великими обсягами даних та забезпечувати швидкий та надійний доступ до них. Необхідність розробки інформаційно-пошукової системи для відділу кадрів випливає з актуальних викликів та потреб сучасного управління персоналом.

Розділ 1
Аналіз предметної області

1.1. Огляд існуючих вебзастосунків

В даному розділі розглянемо вже можливі вебзастосунки, їх недоліки. Провівши це з декількома сайтами буде зрозуміла загальна картина необхідного функціоналу для платформи цієї диплоиної роботи.

1.1.1. LinkedIn

Розпочати можна з такої популярної платформи для пошуку роботи – LinkedIn (www.linkedin.com). Попри те, що вона є доволі розповсюдженою, та всерівно має свої недоліки:

- Залежність від профілю: LinkedIn прив'язаний до профілів користувачів і вимагає від них активно підтримувати та оновлювати свої профілі. Це може створювати певну несправедливість для тих, хто не має часу або можливостей для активного участі на платформі.
- Загроза для конфіденційності: LinkedIn вимагає від користувачів надання особистої інформації, такої як досвід роботи, освіта, контактні дані тощо. Це може створювати потенційну загрозу конфіденційності та приватності користувачів.
- Недостатня відповідність вакансій: Іноді LinkedIn може пропонувати вакансії, які не відповідають повністю критеріям користувача або є неактуальними. Це може спричиняти втрату часу та ресурсів при пошуку роботи.
- Засмічення вмістом: Через широкий обсяг користувачів та велику кількість розміщеного вмісту, LinkedIn може страждати від засмічення та низької якості деяких публікацій. Це може ускладнювати знаходження цінної інформації та взаємодії з потенційними роботодавцями або колегами.
- Обмежена доступність для певних галузей: Хоча LinkedIn є широко використовуваною платформою, вона може бути менш популярною або менш ефективною для деяких специфічних галузей або професій. Деякі роботодавці та професійні спільноти можуть використовувати інші спеціалізовані платформи або мережі для пошуку талантів.

Виходячи з цих проблем, можна зробити висновок - що найбільшим мінусом являється те, що платформа більше схожа на соц-мережу, а не на платформу для пошуку роботи. Велика кількість непотрібного контенту робить створюють асоціацію несерйозності даного застосунку, через що деякі люди зі стажем віддають перевагу іншим вебзастосункам, більш прагматичним та стриманим.

Проте необхідно й виділити позитивні сторону даної платформи. LinkedIn, безумовно, є однією з провідних платформ для професійного мережування та пошуку роботи, через що робить спільноту з однієї категорії людей блищими між собою. Кожен може поділитися у себе в профілі щойно здобутою вищою освітою, закінченням певних курсів, отриманням різних сертифікатів тощо. Знайти для себе необхідний вебінар від різного роду профі, на тему якого цікавишся, або влаштувати свій власний, якщо добре орієнтуєшся в певній сфері, або темі, і хочеш цим поділитись з іншими. До речі, так як платформа є доволі популярною, то зазвичай переважна більшість новачків починають пошуки свого першого заробітку саме на ній.

Для поліпшення платформи можна розглянути впровадження нових функціональностей, посилення контролю над якістю вмісту, поліпшення алгоритмів відповідності вакансій, а також розширення спеціалізованих можливостей для різних галузей. Це допоможе зробити платформу більш ефективною та привабливою для користувачів з різними потребами у пошуку роботи.

1.1.2. Work.ua

Наступним розглянемо такий вебзастосунок, як Work.ua (www.work.ua). Відразу можна примітити з назви те, що даний сайт позиціонує себе з пошуком роботи по території України. На час написання цієї роботи, коли по всій країні росте рівень безробіття у зв’язку з війною, такі платформи як Work.ua здаються рятувальним кругом для народу. Але наразі необхідно визначити які є мінуси цієї платформи:

- Обмежене охоплення: Work.ua фокусується переважно на українському ринку праці, тому може бути менш привабливою для тих, хто шукає роботу за межами України. Це може обмежувати можливості для іноземних працівників або тих, хто мріє про кар'єру за кордоном.
- Обмежена розгалуженість галузей: Work.ua має значний фокус на певних галузях, таких як IT, фінанси, маркетинг тощо, але може бути менш ефективною для пошуку роботи у специфічних галузях або менш популярних сферах.
- Відсутність детальної фільтрації: Платформа може мати обмежені можливості фільтрації вакансій за різними критеріями, такими як рівень заробітної плати, тип зайнятості або розташування. Це може ускладнити точний підбір вакансій, що відповідають потребам користувача.
- Велика конкуренція: У зв'язку зі своєю популярністю Work.ua приваблює велику кількість кандидатів на одну вакансію, що зростає конкуренцію серед претендентів на роботу. Це може ускладнити завоювання бажаної посади та зменшити шанси на успіх.

- Відсутність персоналізації: Платформа може бути менш гнучкою в налаштуванні користувача, адаптуванні до його потреб та наданні індивідуальних рекомендацій. Це може знизити користувальницький досвід та унікальність взаємодії з платформою.
- Обмежена функціональність: Work.ua може не мати деяких додаткових функцій, які присутні на інших платформах для пошуку роботи. Наприклад, може бути обмежена можливість взаємодії з роботодавцями, відгуків про компанії або професійного навчання.
- Недостатня оновленість вакансій: Work.ua може стикатися з проблемою нестачі актуальної інформації про вакансії. Деякі вакансії можуть бути застарілими або вже закритими, що ускладнює пошук роботи та збільшує час, витрачений на перегляд непотрібних оголошень.
- Недостатня забезпеченість інструментами для кандидатів: Work.ua може не мати достатньо розширених інструментів або ресурсів для покращення навичок кандидатів або надання корисних порад та рекомендацій. Це може ускладнити розвиток професійної кар'єри та пошук більш високооплачуваних робочих місць.

Виходячи з цих недоліків можна винести наступне – на період 2023 року ця платформа являється свого роду монополістом у сфері пошуку роботи та через відсутність конкуренції не прагне до розширення або покращення свого функціоналу, який є далеким від ідеалу. Велика конкурентність на кожну вакансію, мала кількість розгалужень по професіям, застаріла версія програмного забезпечення, що робить корисування платформою некомфортною для нових юзерів.

Через велику кількість мінусів у цього вебзастосунку для його покращення необхідно немало змін, аби зробити його конкурентним до інших відомих зарубіжник платформ такого ж типу:

- Розширення охоплення: Work.ua може розширити свої можливості і почати привертати більше іноземних роботодавців та працівників. Це може включати розширення функціональності для пошуку роботи за кордоном, підтримку іноземних мов, партнерство з міжнародними компаніями тощо.
- Розширення галузей: Work.ua може розглянути можливість розширення своєї бази вакансій на менш популярні галузі або специфічні сфери. Це дозволить привернути більше роботодавців та працівників з різних галузей і збільшити розмаїття вакансій на платформі.
- Покращення функціональності фільтрації: Work.ua може вдосконалити свою систему фільтрації вакансій, дозволяючи користувачам точніше налаштовувати критерії пошуку, такі як заробітна плата, тип зайнятості,

розташування тощо. Це допоможе зменшити кількість непотрібних вакансій і полегшить користувачам знаходити відповідні пропозиції.

- Зменшення конкуренції: Work.ua може розглянути можливості поліпшення процесу відбору кандидатів для роботодавців, наприклад, шляхом розширення інструментів для оцінки навичок і презентації кандидатів. Це може зменшити кількість зайвих заявок і забезпечити більш ефективний відбір працівників.
- Персоналізація та індивідуальний підхід: Work.ua може зробити платформу більш гнучкою та інтерактивною, надаючи користувачам більше можливостей для налаштування своїх профілів, відображення рекомендацій та персоналізованої інформації. Це допоможе поліпшити користувацький досвід та зробити платформу більш привабливою для користувачів.
- Постійне оновлення вакансій: Work.ua повинна активно відстежувати та оновлювати вакансії на своєму сайті, щоб забезпечити своїм користувачам актуальну та достовірну інформацію про робочі місця.
- Запровадження нових інструментів та ресурсів: Work.ua може розглянути можливість надання користувачам додаткових інструментів, таких як онлайн-курси, тренінги або поради з пошуку роботи. Це дозволить користувачам поліпшити свої навички та конкурентоспроможність на ринку праці.

Зазначені кроки можуть допомогти Work.ua виправити деякі мінуси та поліпшити користувацький досвід на платформі. Проте, реалізація цих змін вимагатиме відповідних ресурсів та зусиль з боку команди розробників Work.ua.

1.1.3. Dou та Djinni

В цьому блоці розглянемо такі два вебзастосунки, як Dou (www.dou.ua) та Djinni (www.djinni.co). Їх потрібно розгяладати в одному блоці, так як проблеми та їх можливі вирішення однакові.

Dou - це веб-платформа для пошуку роботи в сфері IT та програмування в Україні. Сайт пропонує широкий вибір вакансій із різних галузей IT-індустрії, включаючи розробку програмного забезпечення, тестування, дизайн, аналітику та багато іншого. Він надає можливість розміщувати резюме та шукати вакансії від провідних IT-компаній, стартапів та рекрутингових агентств. Користувачі можуть шукати роботу за різними критеріями, такими як розташування, рівень заробітної плати, тип зайнятості та інші. Сайт також пропонує новини та статті про IT-індустрію, інтерв'ю з експертами та іншу корисну інформацію для професіоналів у сфері IT. Dou.ua є популярним ресурсом серед IT-спеціалістів в Україні, проте варто зазначити, що він фокусується головним чином на роботі в IT-сфері, тому може бути менш привабливим для тих, хто шукає роботу в інших галузях.

Djinni - це популярна веб-платформа для пошуку роботи в сфері IT та програмування в Україні. Сайт спеціалізується на знаходженні вакансій із різних

галузей IT-індустрії, включаючи розробку програмного забезпечення, веб-розробку, аналітику, тестування та інші.

Djinni пропонує зручний інтерфейс, де користувачі можуть створювати профілі, розміщувати резюме та шукати вакансії. Сайт співпрацює з багатьма відомими IT-компаніями та стартапами, які регулярно розміщують вакансії на платформі.

Крім пошуку роботи, Djinni також надає корисні ресурси для IT-спеціалістів, такі як блоги, статті, підручники та подкасти, які допомагають розширювати знання та професійний розвиток. Однак, варто враховувати, що Djinni, схоже на інші платформи, спеціалізується переважно на роботі в галузі IT, тому може бути менш привабливим для тих, хто шукає роботу в інших сферах.

Отже ці два сайти являються однаковими за тим, що вони є вузьконапрямленими та використовуються суто IT-спеціалістами. Найбільша їх проблема в тому, що вони не розповсюджуються на іноземні ринки, а так як розробники завжди прагнуть більшого кар'єрного росту, який можна отримати із закордонними компаніями. Також є проблема з конкуренцією на цих сайтах – подекуди на 1 вакансію приходять біля 800 резюме, що майже унеможливлює шанси на прохід.

1.1.4. Indeed

Даний вебзастосунок, як і LinkedIn, являється одним з найпопулярніших у світі для пошуку заробітку. Indeed (www.indeed.com) надає широкий спектр вакансій з різних галузей та країн, що робить його привабливим для шукачів роботи з усього світу. На ньому користувачі можуть шукати роботу за ключовими словами, місцем розташування, компанією та іншими параметрами. Результати пошуку відображаються з вакансіями з різних джерел, включаючи прямі роботодавців, рекрутингові агентства та інші робочі платформи. Indeed.com також надає користувачам можливість створювати профілі, розміщувати резюме та зберігати вакансії для подальшого перегляду. Крім того, сайт пропонує корисні інструменти, такі як популярність робочих місць, відгуки про компанії, заробітну плату та іншу інформацію, яка може бути корисною при пошуку роботи.

Одним з переваг Indeed.com є його глобальний охоплення, що дозволяє користувачам знаходити роботу як на місцевому ринку, так і за кордоном. Однак, варто враховувати, що через велику кількість вакансій, конкуренція на сайті може бути високою, що може ускладнити завоювання бажаної посади.

На суб'єктивну думку автора, даний сайт є ідеалом для використання його як HR менеджерами, так і тих, хто знаходиться в пошуку нової роботи. На цьому вебзастосунку є все необхідне для комфортного застосування: від різноманітних фільтрів до клієнтів по всьому світі.

1.2. Ключові поняття

Користувач вебзастосунку - це особа або суб'єкт, який використовує або має доступ до вебзастосунку, який є програмним забезпеченням або сервісом, доступним через Інтернет. Користувач вебзастосунку може бути будь-яким індивідом, компанією, організацією або будь-яким іншим суб'єктом, який взаємодіє з вебзастосунком для виконання певних завдань, отримання інформації, здійснення операцій тощо.

Клієнт - У сфері праці, термін "клієнт" відноситься до особи або організації, яка користується послугами працевлаштування, рекрутинговим агентством або іншою установою, що надає послуги по пошуку та підбору персоналу. Клієнтами можуть бути роботодавці, які шукають кваліфікованих працівників для своєї компанії, або індивідуальні кандидати, які шукають роботу і звертаються до посередників для отримання допомоги у пошуку вакансій.

Кандидат - У сфері праці, термін відноситься до особи, яка подає заявку або виражає зацікавленість у працевлаштуванні і претендує на певну вакансію або посаду в організації. Кандидати зазвичай є працесом активного пошуку роботи і виявляють інтерес до конкретних робочих місць, відповідно до своїх кваліфікацій, досвіду та навичок. Кандидати можуть подавати свої резюме або заявки на вакансії безпосередньо до роботодавців, використовуючи онлайн-платформи для пошуку роботи або звертаючись до рекрутингових агентств.

Комунікація між клієнтом та кандидатом, відіграє значну роль у вебзастосунках. Вартує звернути увагу на такі аспекти:

- Повідомлення та сповіщення: Вебзастосунок повинен забезпечувати ефективну систему повідомлень та сповіщень, яка дозволяє кандидату та клієнту легко спілкуватися між собою. Це може включати можливість надсилання текстових повідомлень, електронних листів або нотифікацій через мобільні додатки.
- Профілі користувачів: Вебзастосунок повинен мати функціонал, що дозволяє кандидатам та клієнтам створювати та управляти своїми профілями. Це дозволяє зберігати важливу інформацію про навички, досвід, освіту та інші релевантні дані, які допомагають в розумінні потреб і вимог сторін.
- Пошук та фільтрація: Застосунок повинен мати потужні функції пошуку та фільтрації, які допомагають кандидатам знаходити вакансії, відповідні їхнім критеріям, а клієнтам - знаходити потенційних кандидатів з необхідними характеристиками.
- Календар та планування: Важливо мати інтегрований календар або систему планування, яка дозволяє кандидатам і клієнтам узгоджувати час для зустрічей, співбесід та інших подій. Це допомагає уникнути недорозумінь та покращує організацію комунікації.

- Завантаження документів: Функція завантаження документів дозволяє кандидатам надсилати свої резюме, мотиваційні листи, портфоліо або інші документи, що стосуються процесу відбору. Клієнти також можуть надсилати вимоги щодо вакансій або документи, пов'язані з управлінням персоналом.
- Історія та аналітика: Можливість ведення історії комунікації та зберігання даних про попередні взаємодії між кандидатом і клієнтом є важливою. Аналітичні інструменти можуть надати корисну інформацію щодо ефективності комунікації, трендів та покращень у процесі відбору.

1.3. Формулювання задачі

Виходячи зі всіх вищеперечислених даних, метою кваліфікаційної роботи є розробка вебдодатку з врахуванням таких критеріїв:

- Зручний інтерфейс
- Реєстрація на сайт задля особистого кабінету
- Можливість створювати вакансії
- Подавання заявки на вакансії
- Створення чату для комфортної комунікації
- Різноманітні фільтри для пошуку

Розділ 2
Інструменти розробки платформи

2.1. Вступ

Перед тим як далі буде перелік необхідних нам технологій, фреймворків та програм, необхідно розуміти те, що кожен сайт складається з двох частин: клієнтської (Frontend) та серверної (Backend). Вони є двома основними компонентами, які взаємодіють між собою для забезпечення функціональності та надання послуг користувачам. Нижче наведено опис та порівняння цих двох частин вебдодатку:

Клієнтська частина:

- Опис: Клієнтська частина вебдодатку - це фронтенд або клієнтський інтерфейс, який взаємодіє з користувачем. Вона відповідає за візуальне відображення інформації, взаємодію з користувачем через інтерфейс, обробку даних та передачу їх на серверну частину для обробки.
- Функції: Клієнтська частина забезпечує інтерактивність та користувацький досвід вебдодатку. Вона відповідає за створення та відображення веб-сторінок, форм, кнопок, графічних елементів, анімації тощо. Клієнтська частина також включає логіку клієнта, яка може обробляти події, валідувати дані, виконувати перевірки безпеки та забезпечувати навігацію по додатку.
- Мови та технології: Для розробки клієнтської частини вебдодатку використовуються мови програмування, такі як HTML (HyperText Markup Language), CSS (Cascading Style Sheets) та JavaScript. Технології, такі як фреймворки і бібліотеки JavaScript (наприклад, React, Angular, Vue.js), допомагають спростити розробку клієнтської частини та покращити її продуктивність.

Серверна частина:

- Опис: Серверна частина вебдодатку - це бекенд або серверний компонент, який забезпечує обробку запитів від клієнта, взаємодію з базою даних, бізнес-логікою та іншими зовнішніми сервісами. Вона відповідає за обробку та збереження даних, обробку бізнес-логіки, автентифікацію, авторизацію та інші завдання, які необхідні для функціонування вебдодатку.
- Функції: Серверна частина забезпечує обробку запитів, які надходять від клієнта, і відповідає на них, надаючи необхідні дані та ресурси. Вона може включати механізми маршрутизації, валідації даних, доступу до бази даних, кешування, реалізацію бізнес-логіки та інші компоненти, необхідні для функціонування додатку.
- Мови та технології: Для розробки серверної частини вебдодатку використовуються різні мови програмування, такі як Java, Python, Ruby, PHP,

Node.js тощо. Розробка серверної частини також включає використання фреймворків, таких як Django, Ruby on Rails, Spring, Express.js, для полегшення процесу розробки та забезпечення безпеки та ефективності серверної частини.

Порівняння:

- Залежність: Клієнтська частина залежить від серверної частини, оскільки вона взаємодіє з нею для отримання даних та виконання операцій. Серверна частина може бути незалежною, оскільки вона може обробляти запити та здійснювати свою функціональність навіть без клієнта.
- Відповідальності: Клієнтська частина відповідає за візуальне відображення, інтерактивність та користувацький досвід, тоді як серверна частина забезпечує обробку даних, бізнес-логіку та інші серверні функції.
- Технології: Клієнтська та серверна частини можуть використовувати різні технології та мови програмування, в залежності від вимог проекту та вибору розробників. Проте, важливо, щоб вони могли взаємодіяти між собою за допомогою стандартних протоколів, таких як HTTP або WebSockets.

В цілому, клієнтська та серверна частини вебдодатку співпрацюють, щоб забезпечити повноцінний та функціональний вебдосвід для користувачів. Клієнтська частина забезпечує візуальне відображення та взаємодію з користувачем, тоді як серверна частина відповідає за обробку даних та надання необхідних ресурсів.

2.2. Клієнтська частина

Angular - це платформа для розробки застосунків та для створення ефективних і складних односторінкових додатків, що була розроблена компанією “Google” на базі фреймворку “AngularJS”. Angular не є оновленою версією AngularJS, а вважається самостійним фреймворком з іншими цілями, методами та філософією.

Angular націлений на розробку SPA - додатків (Single Page Application), тобто односторінкових сервісів, які стають все більш популярними з розвитком малого бізнесу, зацікавленості у підприємців до популяризації свого бренду або виробництва.

Angular забезпечує двостороннє зв'язування, що дозволяє динамічно змінювати дані в одному місці інтерфейсу при зміні даних моделі в іншому.

Однією з ключових особливостей Angular є те, що він використовує TypeScript в якості мови програмування, але не обмежує одним лише TypeScript. Фреймворк дозволяє також використовувати такі мови, як JavaScript або Dart. Однак TypeScript

все таки є основною мовою для Angular. **TypeScript**. Фреймворк дозволяє також використовувати такі мови, як JavaScript або Dart. Однак TypeScript все таки є основною мовою для Angular.

Компоненти - це будівельні блоки, з яких складається програма. Компонент включає клас TypeScript з декоратором @Component (), шаблон HTML та стилі. Декоратор @Component () визначає таку інформацію, специфічну для Angular:

- Селектор CSS, який визначає, як компонент використовується у шаблонах. Елементи HTML - елементів у шаблоні, які відповідають цьому селектору, стають примірниками компонента.
- Шаблон HTML, який вказує Angular як відтворювати компонент.
- Необов'язковий набір стилів CSS, які визначають зовнішній вигляд HTML - елементів шаблону.[1]

Приклад компоненту:

```
import { Component } from '@angular/core';
@Component({
  selector: 'hello-world',
  template: `
    <h2>Hello World</h2>
    <p>This is my first component!</p>
    `,
})
export class HelloWorldComponent {
  // програмний код
}
```

Щоб використовувати цей компонент, слід вписати в шаблон даний код:

```
<hello-world></hello-world>
```

Коли Angular рендерить цей компонент, результат DOM виглядає так:

```
<hello-world>
 <h2>Hello World</h2>
 <p>This is my first component!</p>
</hello-world>[1]
```

Компонентна модель Angular пропонує міцну інкапсуляцію та інтуїтивно зрозумілу структуру додатків. Компоненти також роблять вашу програму

безболісною для модульного тестування та можуть покращити загальну читаність вашого коду.

JavaScript (також відомий як ECMAScript) почав своє життя як проста мова сценаріїв для браузерів. На момент винайдення його передбачалося використовувати для коротких фрагментів коду, вбудованого у веб - сторінку - написання кількох десятків рядків коду було б дещо незвичним. Завдяки цьому ранні веб - браузери виконували такий код досить повільно. Однак з часом JS став все більш популярним, і веб - розробники почали використовувати його для створення інтерактивного досвіду.

Розробники веб - браузерів відреагували на це збільшення використання JS, оптимізацією своїх механізмів виконання (динамічна компіляція) та розширивши можливості, які можна було б з цим зробити (додавши API), що, у свою чергу, змусило веб - розробників використовувати їх ще більше. На сучасних веб - сайтах у вашому браузері часто працюють програми, які охоплюють сотні тисяч рядків коду. Це тривале і поступове зростання «Інтернету», починаючи як просту мережу статичних сторінок і перетворюючись на платформу для різноманітних додатків усіх видів.[2]

Більш того, JS став досить популярним для того, щоб його можна було використовувати поза контекстом браузерів, таких як реалізація серверів JS за допомогою node.js. Природа JS "бігти куди завгодно" робить її привабливим вибором для крос - платформенної розробки. Сьогодні є багато розробників, які використовують лише JavaScript для програмування всього свого стека.

JavaScript також дозволяє отримати доступ до властивостей, яких немає:

```
const obj = {width:10, height:15};
// Why is this NaN? Spelling is hard!
const area = obj.width * obj.heigth;
```

Виявлення помилок у коді без його запуску називається статичною перевіркою. Визначення того, що є помилкою, а що не ґрунтується на типах оперованих значень, відоме як статична перевірка типу.

TypeScript перед виконанням перевіряє програму на наявність помилок і робить це на основі видів значень, це перевірка статичного типу. Наприклад, останній приклад вище містить помилку через тип obj. Ось знайдена помилка TypeScript:

```
const obj = {width:10, height:15};
const area = obj.width * obj.heigth;
```

Property 'height' does not exist on type '{ width : number, heigth : number; }'. Did you mean 'height'?

TypeScript незвично відноситься до JavaScript. TypeScript пропонує всі функції JavaScript та додатковий шар поверх них: систему типів TypeScript.

Наприклад, JavaScript надає такі примітивні типи, як число, рядок, символ, але не перевіряє, чи їх правильно призначали. Це робить TypeScript.[1]

Це означає, що ваш існуючий робочий код JavaScript також є кодом TypeScript. Основною перевагою TypeScript є те, що він може виділяти несподівану поведінку у вашому коді, зменшуючи ймовірність помилок.

Dependency injection. Ін'єкція залежностей дозволяє оголошувати залежності ваших класів TypeScript, не дбаючи про їх створення. Натомість Angular обробляє інсталяцію за вас. Цей підхід до дизайну додатків дозволяє писати більш перевірений та гнучкий код. Навіть якщо розуміння введення залежностей не є критичним для початку використання Angular, я настійно рекомендуємо його як найкращу практику, і багато аспектів Angular певною мірою цим користуються.

Щоб проілюструвати, як працює введення залежностей, розглянемо наступний приклад. Перший файл, logger.service.ts, визначає клас Logger. Цей клас містить функцію writeCount, яка реєструє номер на консолі.[5]

Далі файл hello-world-di.component.ts визначає компонент Angular. Цей компонент містить кнопку, яка використовує функцію writeCount класу Logger. Щоб отримати доступ до цієї функції, служба Logger вводиться в клас HelloWorldDI шляхом додавання приватного logger: Logger до конструктора.

```
import { Component } from '@angular/core';
import { Logger } from '../logger.service';
@Component({
  selector: 'hello-world-di',
  templateUrl: './hello-world-di.component.html'
})
export class HelloWorldDependencyInjectionComponent  {
  count = 0;
  constructor(private logger: Logger) {
  }
  onLogMe() {
    this.logger.writeCount(this.count);
    this.count++;
  }
}
```

Angular CLI - це найшвидший, простий та рекомендований розробниками спосіб проектування та реалізації програм, що базуються на фреймворку Angular.

Команда ng new створює папку робочого простору Angular та генерує новий скелет програми. Робоча область може містити кілька додатків і бібліотек.

Початкова програма, створена командою ng new, знаходиться на верхньому рівні робочої області. Коли ви створюєте додаткову програму або бібліотеку в робочій області, вона переходить у проект/ підпапку.

Щойно створений додаток містить вихідні файли для кореневого модуля з кореневим компонентом та шаблоном. Кожна програма має папку src, яка містить логіку, дані та ресурси.

Ви можете безпосередньо редагувати створені файли або додавати та змінювати їх за допомогою команд CLI. Використовуйте команду ng generated, щоб додати нові файли для додаткових компонентів та служб, а також код для нових каналів, директив тощо. Такі команди, як додавання та генерування, які створюють або працюють над програмами та бібліотеками, повинні виконуватися з робочої області чи папки проекту.

Angular CLI робить ряд завдань простими та надає можливість повторного використання коду. Наприклад:

ng build	Компілює додаток Angular у ідний каталог
ng serve	Створює та обслуговує вашу програму, слідкуючи за змінами у файлах
ng generate	Створює або змінює файли на основі схеми
ng test	Запускає модульні тести для певного проекту
ng e2e	Створює та обслуговує додаток ular, а потім запускає процес повного ування додатку

Таб. 1. Angular CLI команди

Angular Service Worker. Завдяки вбудованій підтримці PWA Angular Service Worker та Angular CLI розробляти нативні мобільні додатки стало набагато легше та зручніше. Service Worker доповнює традиційну модель розгортання проектів в Інтернеті та розширює можливості програм, щоб забезпечити зручність та досвід використання додатку користувачем із надійністю та продуктивністю нарівні з нативно - декларованим кодом. Додавання сервісного працівника до програми Angular - це один із кроків перетворення програми в прогресивну веб - програму (також відому як PWA).[4]

Працівники служби функціонують як мережевий проксі. Вони перехоплюють усі вихідні HTTP - запити, зроблені програмою, і можуть обирати, як на них відповідати. Наприклад, вони можуть запитувати локальний кеш і доставляти кешовану відповідь, якщо вона доступна. Проксі - сервер не обмежується

запитами, зробленими через програмні API, наприклад, fetch; він також містить ресурси, на які посилається HTML, і навіть початковий запит до index.html. Таким чином, кешування на основі сервісних працівників повністю програмується і не покладається на визначені сервером заголовки кешування.[5]

На відміну від інших скриптів, що входять в склад програми, таких як Angular application bundle, service worker зберігає користувацькі запити після того, як користувач закриє вкладку. Наступного разу, коли браузер завантажить додаток, service worker завантажиться першим і може перехопити кожен запит на ресурси для завантаження програми. Якщо сервісний працівник призначений для цього, він може повністю задовольнити завантаження програми, незалежно від наявності підключення до мережі Інтернет.

2.3. Серверна частина

Backend частина буде реалізована на технології **.NET Core**. Реалізюється вона за допомогою бібліотек на мові **C#**, та була випущена компанією Microsoft.

Це є кросс-платформерна технологія, що допомагає розроблювати додатки нею на ПК, телефони та інші платформи без великого пересування коду. Також потрібно згадати що вона славиться своєю швидкодією та ефективністю завдяки оптимізованому середовищу. Багато розробників по всьому світу використовують .NET Core, через що в інтернеті можна знайти значну частину готових рішень від інших людей.

База данних у проекті буде реалізована за допомогою технології **PostgreSQL**. Це потужна об'єктно-реляційна система баз даних з відкритим вихідним кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають та масштабують найскладніші навантаження на дані. Витоки PostgreSQL сягають 1986 року в рамках проекту POSTGRES в Каліфорнійському університеті в Берклі і мають понад 30 років активного розвитку на основній платформі.

PostgreSQL заслужив міцну репутацію завдяки своїй перевіреній архітектурі, надійності, цілісності даних, надійному набору функцій, розширюваності та відданості спільноти з відкритим кодом, що стоїть за програмним забезпеченням, щоб послідовно пропонувати ефективні та інноваційні рішення. PostgreSQL працює на всіх основних операційних системах, сумісний з ACID з 2001 року і має потужні доповнення, такі як популярний розширювач геопросторових баз даних PostGIS. Не дивно, що PostgreSQL став реляційною базою даних з відкритим вихідним кодом для багатьох людей та організацій. [3]

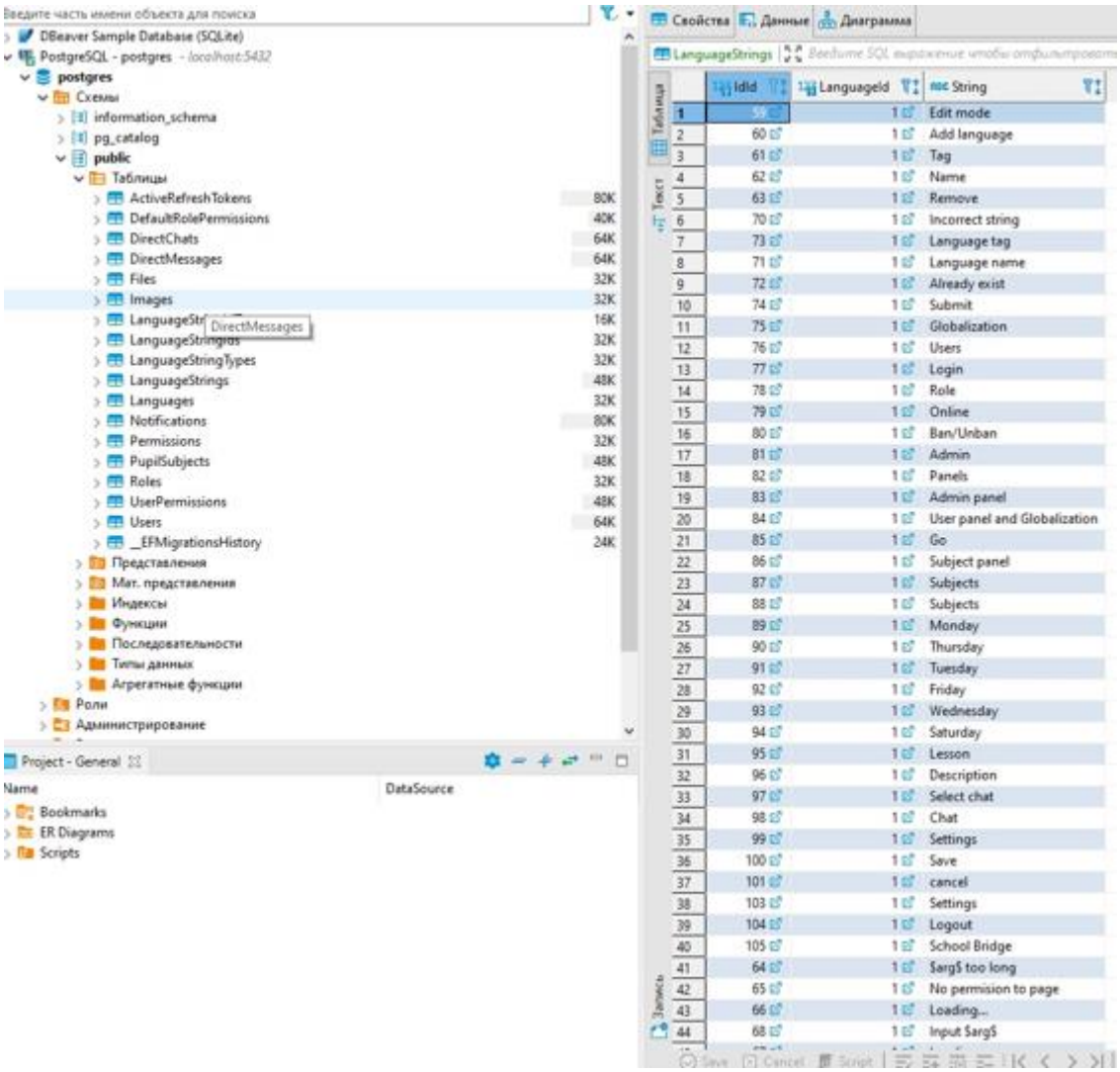

Рис. 1. База даних PostgreSQL

Звісно, так як в проекті буде реалізована реєстрація та авторизація, то не потрібно забувати про шифрування токенів. Це можна зробити за допомогою токенів. **JSON Web Token (JWT)** - це відкритий стандарт (RFC 7519), який визначає компактний та автономний спосіб безпечної передачі інформації між сторонами як об’єкт JSON. Цю інформацію можна перевірити та довірити, оскільки вона підписана цифровим способом. JWT можна підписувати за допомогою секретного (з алгоритмом HMAC) або пари відкритого/закритого ключів за допомогою RSA або ECDSA.

Хоча JWT - токени можуть бути зашифровані для забезпечення таємниці між сторонами, ми зосередимось на підписаних маркерах. Підписані маркери можуть перевіряти цілісність вимог, що містяться в ньому, тоді як зашифровані маркери приховують ці претензії від інших сторін. Коли маркери підписуються за допомогою пар відкритого/закритого ключів, підпис також засвідчує, що підписала його лише сторона, яка володіє закритим ключем.[3]

Актуальність використання JWT маркерів у проекті:

Авторизація: після входу користувача кожен наступний запит включатиме JWT, що дозволяє користувачеві отримувати доступ до маршрутів, служб та ресурсів, дозволених цим маркером. Єдиний вхід - це функція, яка сьогодні широко використовує JWT через її невеликі накладні витрати та здатність легко використовуватись у різних доменах.

Обмін інформацією: веб -маркери JSON - це хороший спосіб безпечної передачі інформації між сторонами. Оскільки JWT можна підписувати - наприклад, за допомогою пар відкритих/приватних ключів - ви можете бути впевнені, що відправники - це ті, кого вони називають. Крім того, оскільки підпис обчислюється за допомогою заголовка та корисного навантаження, ви також можете перевірити, чи вміст не підроблено.[4]

Приклад налаштувань JWT:

```
"JwtSettings": {
        "Key":
"IWANNABETHEBOSHANDHAVEASQUAREFORMNOTTRIANGLE",
        "RefreshKey":
"IWANNABETHEBOSHANDHAVEASQUAREFORMNOTTRIANGLE2",
        "Issuer": "https://greenp.space",
        "ExpireMinuts": 15,
        "RefreshExpireDays": 30,
        "RefreshRemoveDays": 1
    },
    "PermanentConnectionService": {
        "PermanentKey":
"IWANNABETHEBOSHANDHAVEASQUAREFORMNOTTRIANGLE3",
        "PermanentExpireMinuts": 15
    },
    "OnlineService": {
        "OnlineKey":
"IWANNABETHEBOSHANDHAVEASQUAREFORMNOTTRIANGLE5"
    },
    "RegistrationService": {
        "RegistrationKey":
"IWANNABETHEBOSHANDHAVEASQUAREFORMNOTTRIANGLE4",
        "RegistrationExpireDays": 1,
        "ChangePasswordExpireDays":  1
    }
```

Розділ 3
Етапи розробки

3.1. Створення серверної частини

Задля прозорої архітектури серверної частини було прийнято рішення розбити його на декілька підпроектів:

- API – головна частина, яка здійснює запуск всього серверу. В ньому знаходяться всі контроллери, конфігураційні файли для зв'язку з базою даних, налаштування ключів для JWT токенів і тд.
- DataAccess – підпроект, що слугує свого роду "мозком". Тут розміщені всі налаштування об'єктів для бази даних, міграції та репозиторії.
- Domain та Helpers – частини, які виконують роль сервісів. Відмінність між ними полягає тільки в тому, що перший зберігає всю першочергову логіку, яка буде використовуватись при модифікаціях об'єктів та моделей, а другий слугує допоміжним, в якому записані другорядні сервіси по типу логування, хешування паролів, відправки електронних листів, моніторингу онлайну відповідно.

Що ж для початку необхідно скачати до проекту пакети з Рис. 3.1. для подальшої розробки.

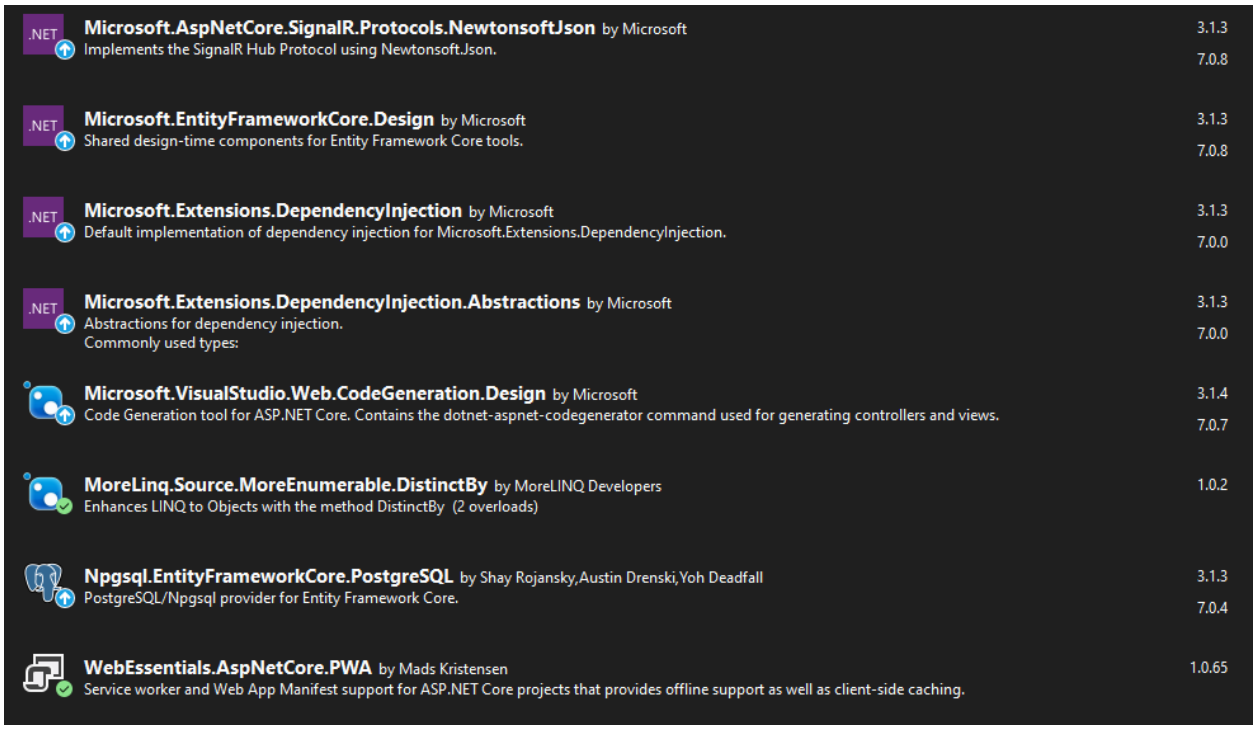

Рис 3.1. Необхідні NuGet пакети для подальшої роботи

Серед списку ви можете помітити Entity Framework. Саме з нього розпочнеться робота. Створюємо інтерфейс класу **GenericRepository** з оголошеннями методів та їх реалізація [6]. Далі потрібно створити всі об'єкти для

моделі (Рис. 3.2.), з якими ми будемо в подальшому працювати через репозиторій, полегшуючи собі роботу з базою даних.

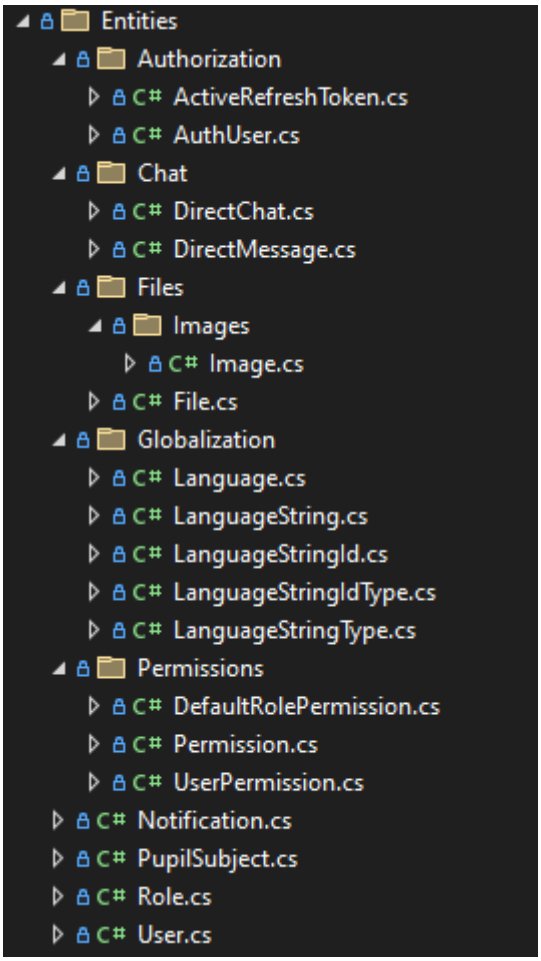

Рис. 3.2.

На даному етапі ми готуємось до того, щоб створити нашу базу даних. Для цього потрібно налаштувати конфігураційний файл проекту.

```
"ConnectionStrings": {
    "mymssql": "Data Source=localhost;Database=GreenPDB;Integrated
Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;Mu
ltiSubnetFailover=False;MultipleActiveResultSets=True",
    "mssql":
"Server=(localdb)\\mssqllocaldb;Database=GreenPDB;Trusted_Connection=True;",
    "postgres":
"Host=91.238.103.107;Port=5432;Database=postgres;Username=postgres;Password=greenp00
7post",
    "mypostgres":
"Host=127.0.0.1;Port=5432;Database=postgres;Username=postgres;Password=345123"
  }
```

Та прописати підключення у файлі Startup.cs наступним чином:

```
  // Db Context
            services.AddDbContext<DbContext, ApplicationContext>(opt =>

opt.UseNpgsql(_configuration.GetSection("ConnectionStrings")["mypostgres"]),
optionsLifetime: ServiceLifetime.Singleton);
```

Тепер необхідно зробити перші міграції задля переносу наших моделей в бд. Для цього потрібно перейти в спеціально консоль за шляхом **Tools -> NuGet Package Manager -> Package Manager Console,** де прописуємо команди:

1. Add-Migration <migration name>
2. Update-Database

Після цих дій ми можемо перевірити базу даних, чи створились таблиці. Після підключення ви можете спостерігати налаштовану та готову до роботи бд (Рис 3.3.)

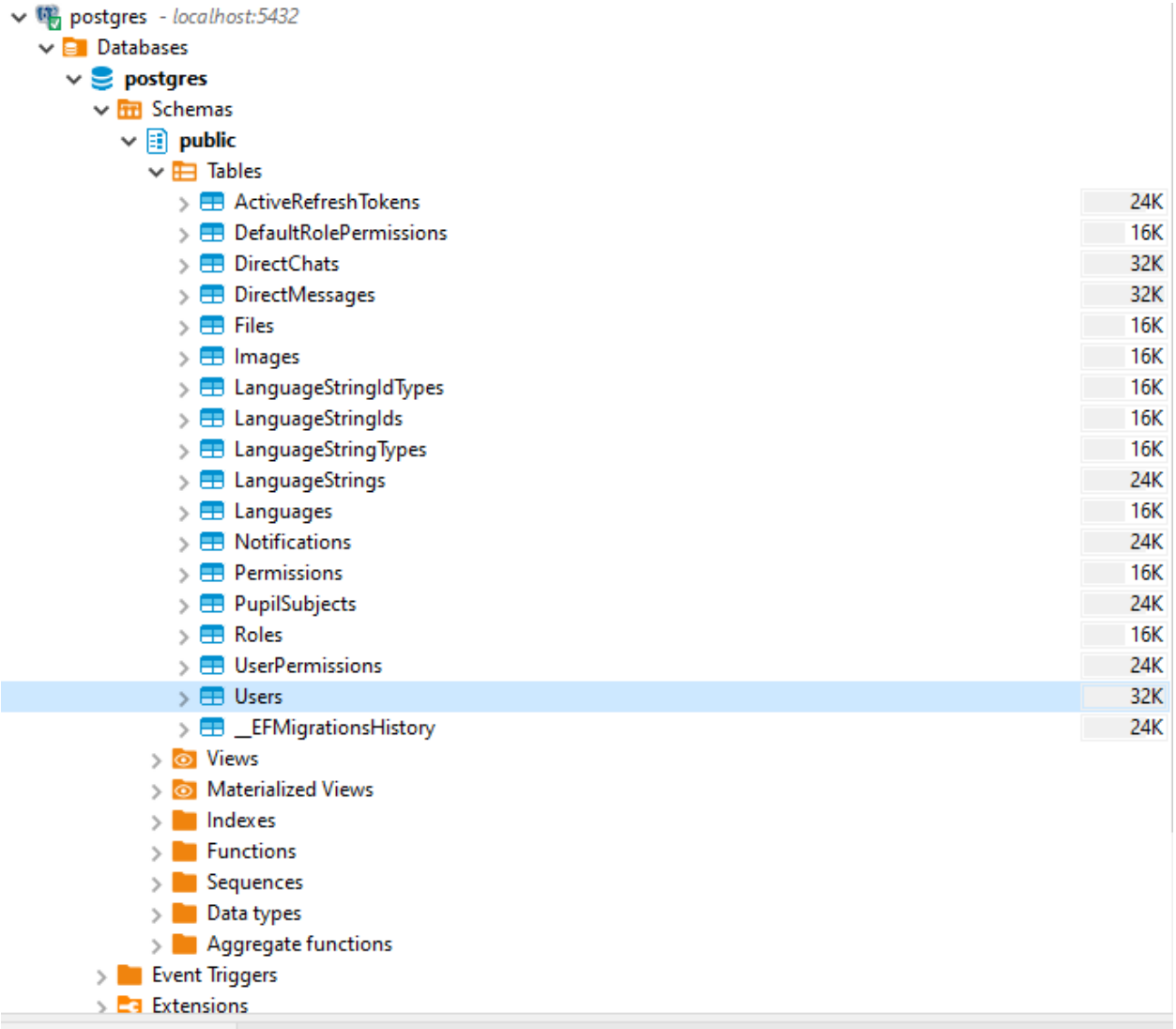

Рис 3.3. Створені таблиці в PostgreSQL

Після пройдених операцій пора взятися за класи, відповідаючих за логіку додатку, а саме – сервіси. В першу чергу необхідні ті, що відповідають за авторизацію:

- EmailService – сервіс на основі бібліотеки Smtp, створений для відправлення електронних листів користувачам, які хочуть зареєструватись.

- JwtTokenService та PermanentConnectionService – сервіси для створення Jwt токенів, які будуть відповідати за те, щоб користувачів не розлогінювало після перезапуску сторінки.
- LoginService та RegistrationService – містять методи для авторизації та реєстрації користувачів. Вони будуть зв’язуватись з сервісами токенів та записувати їх відповідно для кожного юзера до бази даних.
- PermissionService, RoleService, ValidatingService та UserConnectionService – допоміжні класи для видачі користувачам доступи різного роду, ролей а також валідації даних.

Після цього нам потрібні контроллери LoginController, ErrorsController, RegisterController, UserController (Рис. 3.4.). Вони будуть містити виклики аналогічним їх назвам сервісів, через які будуть зв’язуватись серверна та клієнтська частини за допомогою запросів HTTP.

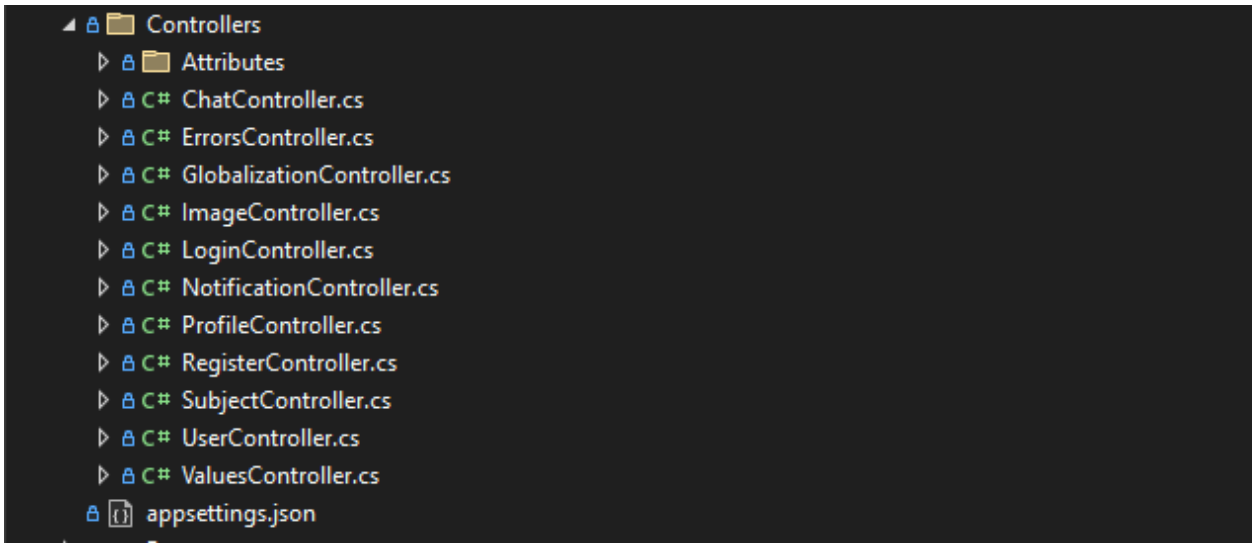

Рис. 3.4.

Так як додаток повинен бути міжнародним – йому необхідна функція додавання інших мов, їх редагування. На серверній частині необхідно підготувати функціонал, який буде приймати зміни з клієнтської частини та записувати їх до бази даних, а також не прописати мови, які будуть записані до бд з першого запуску – українська та англійська. Для цієї частини був створений окремий контроллер GlobalizationController, сервіс LanguageStringService та моделі Language.cs, LanguageString.cs, LanguageStringId.cs, LanguageStringIdType.cs, LanguageStringType.cs.

3.2. Процес розробки клієнтської частини

Перед тим як почати працювати на стороні фронтенду – необхідно завантажити всі пакети технологій, без яких вебзастосунок не буде працювати.

І почати необхідно зі завантаження Node.js. з офінійного сайту (www.nodejs.org) версії, яка підходитиме нашій платформі. Після завантаження та встановлення необхідно запустити Node.js Command Prompt та перейти в директорію проекту. Якщо на даний момент в робочому середовищі немає Angular-у, то потрібно його отримати шляхом прописання команди “npm i -g @angular/cli”. Після того створюємо проект та скачуємо/оновлюємо всі модулі пакету командою “npm i”. Після цих дій залишається лише запустити клієнтську частину командою “ng serve” та починати роботу.

Створюємо перші компоненти Home, Login, Register, Email-Register, з яких буде розпочинати працювати сайт. Кожен з цих елементів містить три типи файлів у собі, а саме: .html та .css – файли, що регулюються мовою розмітки, для того, щоб задавати вигляду кожній компоненті вебзастосунку, та .ts – для прописання логіки кожному елементу мовою TypeScript.

Необхідно також відразу налаштувати роути застосунку для можливості переходу між сторінками без його перезагрузки. Таку можливість ми маємо завдяки тому, що Angular являється фреймфорком SPA – Single Page Application.

Після налаштування файлу app-routing.module.ts маємо вид:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule, UrlSegment, UrlSegmentGroup, UrlMatchResult } from
'@angular/router';
import { LoginRegisterGuard } from './Guard/login-register.guard';
import { StartComponent } from './Modules/start/main/start.component';
import { PanelComponent } from './Modules/panel/main/panel.component';
import { LoginnedGuard } from './Guard/loginned.guard';
import { DefaultComponent } from './Components/default/default.component';


export const routes: Routes = [
  { path: 'start',
    component: StartComponent,
    canActivate: [LoginRegisterGuard],
    canLoad: [LoginRegisterGuard],
    runGuardsAndResolvers: 'always',
    loadChildren: () => {
      return import('./Modules/start/start.module').then(m => m.StartModule)
    },
  },
  { path: 'panel',
    component: PanelComponent,
    canActivate: [LoginnedGuard],
    canLoad: [LoginnedGuard],
    runGuardsAndResolvers: 'always',
    loadChildren: () => import('./Modules/panel/panel.module').then(m =>
m.PanelModule),
```

```
  },
  { path: '',
    pathMatch: 'full',
    component: DefaultComponent,
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Також ви могли б помітити що на деякі компоненти вже працюють **Guard**-и. Це бібліотека, що допомагає закрити на потрібні компоненти доступ користувачам, які не мають необхідних прав(Рис. 3.5), або не є авторизованими (Рис. 3.6). У разі непроходження через цю бібліотеку, користувач буде сповіщений про це.

```
export class LoginRegisterGuard implements CanActivate, CanActivateChild, CanLoad {
  constructor(
    private userService: UserService,
    private toastrService: Toaster,
    private router: Router,
    private _guardService: GuardService,
    private _gbsService: GlobalizationStringService
  ) {}

  canActivateChild(
    childRoute: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ) {
    return this.canActivate(childRoute, state);
  }

  canLoad(route: Route): boolean {
    return this.canActivate(null, null);
  }s

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
    if (!this._guardService.getState()) return true;
    if (!this.userService.user) return true;

    this.toastrService.open(this._gbsService.getStringObs("no-perm-page"), {type: 'danger'});
    this.router.navigateByUrl("/panel");
    return false;
  }
}
```

Рис. 3.5 Guard-и для юзерів без необхідних прав доступу

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router, CanActivateChild, CanLoad, Route } from '@angular/router';
import { UserService } from '../Services/user.service';
//import { ToastrService } from 'ngx-toastr';
import { GuardService } from '../Services/guard.service';
import { Toaster } from '../Modules/ngx-toast-notifications';
import { GlobalizationStringService } from '../Modules/globalization/Services/globalization-string.service';

@Injectable({ providedIn: 'root' })
export class LoginnedGuard implements CanActivate, CanActivateChild, CanLoad {
    constructor(private userService: UserService,
                private toastrService: Toaster,
                private router: Router,
                private _guardService: GuardService,
                private _gbsService: GlobalizationStringService) { }

    canActivateChild(childRoute: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
        return this.canActivate(childRoute, state);
    }

    canLoad(route: Route): boolean {
      return this.canActivate(null, null);
    }

    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
        if (!this._guardService.getState())
            return true;
        if (this.userService.user)
            return true;
        this.toastrService.open(this._gbsService.getStringObs('no-perm-page'), {type: 'danger'});
        this.router.navigateByUrl('/start?returnUrl=' + encodeURI(window.location.pathname));
        return false;
    }
}
```

Рис. 3.6. Guard-и для неавторизованих юзерів

Далі налаштовуємо елементи Login та Register. Для зручного витягування даних зі сторінки будемо користуватись NgxForm-ами, як на прикладу у логіна на Рис. 3.7:

```
export class LoginComponent extends OnUnsubscribe implements OnInit {
  public returnUrl: string = null;
  public form: NgxFormModel = new NgxFormModel("form", ["login", { name: "password", type: "password" }]);

  constructor(private authService: LoginService,
    private router: Router,
    private route: ActivatedRoute,
    private userPermisionService: UserPermissionService) {
    super();
  }
```

Рис. 3.7. Приклад використання NgxForm на компоненті сторінки логіну

Тепер можна зробити глобалізацію на клієнтській стороні. Це краще зробити чим раніше, тому що з цього моменту потрібно буде у всіх файлах html дописувати власний префікс, щоб система розуміла де знаходяться написи, що потрібно записувати на декількох мовах. Це ми будемо помічати префіксом **dbString**. До речі, в цій частині нам необхідно використовувати реактивне програмування з бібліоткекою **rxjs**, для того, щоб в подальшому нам було зручно змінювати з адмін панелі написи вручну. Код реалізації першого сервісу з цього функціоналу показано на Рис. 3.8.

```typescript
import { Injectable } from '@angular/core';
import { MyLocalStorageService } from 'src/app/Services/my-local-storage.service';
import { HttpGlobalizationService } from './http-globalization.service';
import { GlobalizationInfoService } from './globalization-info.service';
import { GlobalizationStringService } from './globalization-string.service';
import { tap } from 'rxjs/operators';
import { Observable } from 'rxjs';
import { GlobalizationInfo } from '../Models/globalization-info.model';

@Injectable()
export class GlobalizationService {

    constructor(private _localStorage: MyLocalStorageService,
                private _httpGbService: HttpGlobalizationService,
                private _gbiService: GlobalizationInfoService,
                private _gbsService: GlobalizationStringService){

        this._httpGbService.getInfo().subscribe(x => {
            if (this._gbiService.info && x.baseUpdateId != this._gbiService.info.baseUpdateId)
                this._gbsService.clearAllData();
            this._gbiService.info = x;
        });
    }

    public localSetInfo(info: GlobalizationInfo){
        if (info.baseUpdateId != this._gbiService.info.baseUpdateId)
            this._gbsService.clearAllData();
        this._gbiService.info = info;
    }

    public changeLanguage(lang: string = ""): Observable<GlobalizationInfo>{
        return this._httpGbService.getInfo(lang).pipe(tap(x => {
            if (x.baseUpdateId != this._gbiService.info.baseUpdateId)
                this._gbsService.clearAllData();
            this._gbiService.info = x;
            this._gbsService.loadAllNoLoadedString();
        }));
    }
}

```

Рис. 3.8. Реалізація сервісу допоміжного сервісу по глобалізації

Також необхідний сервіс, який буде відправляти зміни на серверну частину для зберігання їх у базі даних. Цим буде займатись сервіс globalization-info.service. Частину його реалізації зображено на Рис. 3.9 та Рис. 3.10. Він буде приймати необхідну інформацію та відправляти через Http протоколи, а також надавати необхідну інформацію при звернені до нього.

```typescript
export class GlobalizationInfoService {
  public readonly loading: IsLoading = new IsLoading();

  private _info: BehaviorSubject<GlobalizationInfo> = new BehaviorSubject<
    GlobalizationInfo
  >(null);
  private _infoObs: Observable<
    GlobalizationInfo
  > = this._info.asObservable().pipe(
    delayWhen((x) => (this.loading.status ? this.loading.event.pipe(filter(x => !x)) : of())),
    map((x) => this.info)
  );
```

Рис. 3.9 Реалізація класу globalization-info.service

```
public get infoObs(): Observable<GlobalizationInfo> {
  return this._infoObs;
}

public get info(): GlobalizationInfo {
  return this._info.value;
}
public set info(val: GlobalizationInfo) {
  this._localStorage.write("gbinfo", val);
  this._info.next(val);
}

constructor(private _localStorage: MyLocalStorageService) {
  if (this._localStorage.isIssetKey("gbinfo")) {
    this._info.next(this._localStorage.read("gbinfo"));
  }
}
```

Рис. 3.10 Друга частина реалізації globalization-info.service

Тепер сервіс, який буде працювати з сервісом вище – http-globalization.service. Він буде слугувати тим, що зв’язує вже існуючі класи для цього функціоналу. Варто зауважити що це все можна було б об’єднати в один файл, але для того щоб зробити код зрозумілим, то необхідно дотримуватись розгалуженню функціоналу. Реалізацію цього сервісу можна побачити на Рис. 3.11 та Рис. 3.12.

```
@Injectable()
export class HttpGlobalizationService {
  private _ser: Service;

  constructor(
    private _baseService: BaseService,
    private _infoService: GlobalizationInfoService
  ) {
    this._ser = apiConfig["globalization"];
  }

  public getInfo(lang: string = ""): Observable<GlobalizationInfo> {
    this._infoService.loading.status = true;

    return this._baseService
      .send<GlobalizationInfo>(
        this._ser,
        "info",
        null,
        this.createHeaders(lang)
      )
      .pipe(finalize(() => this._infoService.loading.status = false));
  }
```

Рис. 3.11. Частина реалізації зв’язуючого класу глобалізації з іншими

```
public updateBaseUpdateId(): Observable<GlobalizationInfo> {
  return this._baseService.send<GlobalizationInfo>(
    this._ser,
    "update-baseupdateid",
    null,
    this.createHeaders()
  );
}

public setString(name: string, str: string): Observable<any> {
  return this._baseService.send<any>(
    this._ser,
    "add-or-upd-string",
    { name: name, string: str },
    this.createHeaders()
  );
}

public getStrings(strs: string[]): Observable<Record<string, string>> {
  return this._baseService.send<Record<string, string>>(
    this._ser,
    "strings",
    { strings: strs },
    this.createHeaders()
  );
}

public addLanguage(abbName: string, fullName: string): Observable<any> {
  return this._baseService.send<any>(this._ser, "add-language", {
    abbName,
    fullName,
  });
}

public removeLanguage(abbName: string): Observable<any> {
  return this._baseService.get(this._ser, "remove-language", {
    params: { abbName: abbName },
  });
}
```

Рис. 3.12. Інша частина реалізації класу http-globalization.service

Розділ 4

Реалізація продукту

При переході за посиланням localhost:4200 користувача зустрічає сторінка з можливістю реєстрації акаунту, входу та зміни мови без перезавантаження сторінки Рис. 4. 1. Для відображення даної сторінки використовуємо імпорт локалізації, анімації angular, його ж роутінг. Також посередина знаходиться кнопка для запросу на отримання консультації.

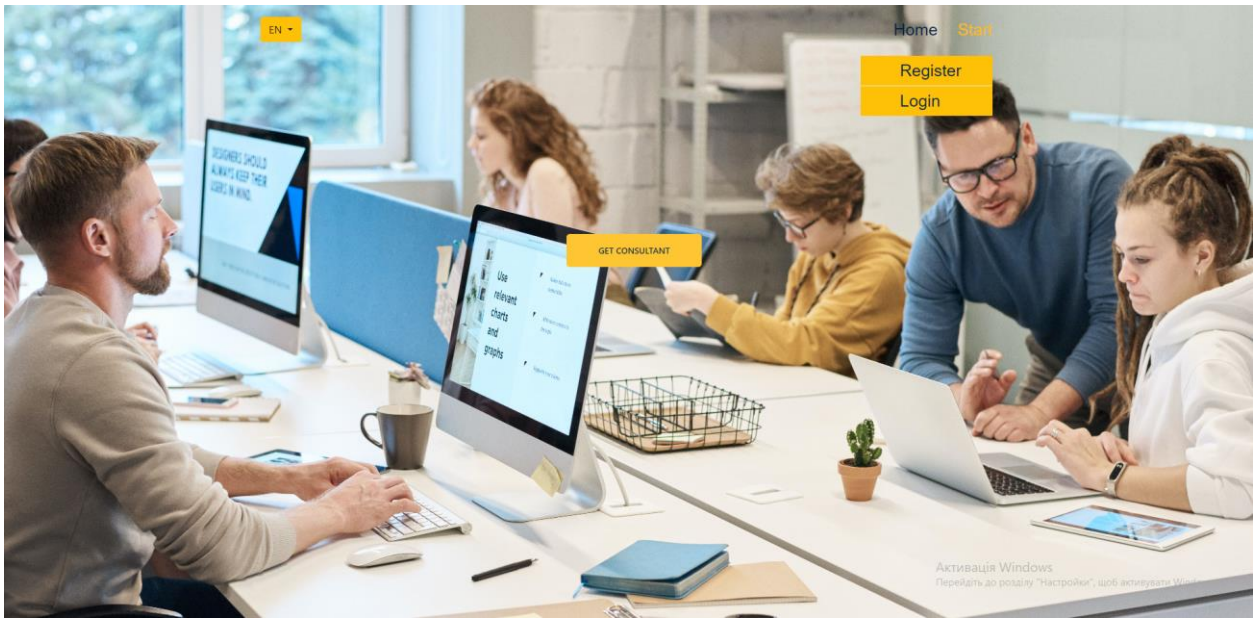

Рис 4.1. Перша сторінка

Просроливши трохи вниз, ви можете побачити доступні на сьогодні відкриті вакансії, які доступні у клієнта.

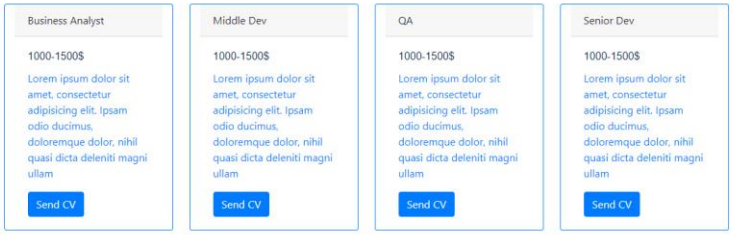

Рис 4.2. Друга частина головної сторінки

Попробувавши змінити мову на українську, ви отримаєте наступну картину (Рис. 4.3). Всі написи змінилися на “-NONE-”. Це означає, що на даний момент адміністрація сайте не добавила цю мову, цим ми займемось далі при перегляді адміністративної панелі.

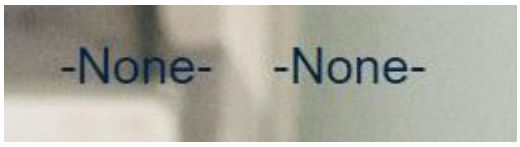

Рис. 4.3. Приклад написів, коли вибрана мова не була добавлена адміністрацією.

Прововжимо розгляд сторінки перейшовшт по кнопці “Register”. скрипт відправляє дані на бек-енд, де відпрацьовується і повертає сторінку реєстрації (Рис. 4.4.).

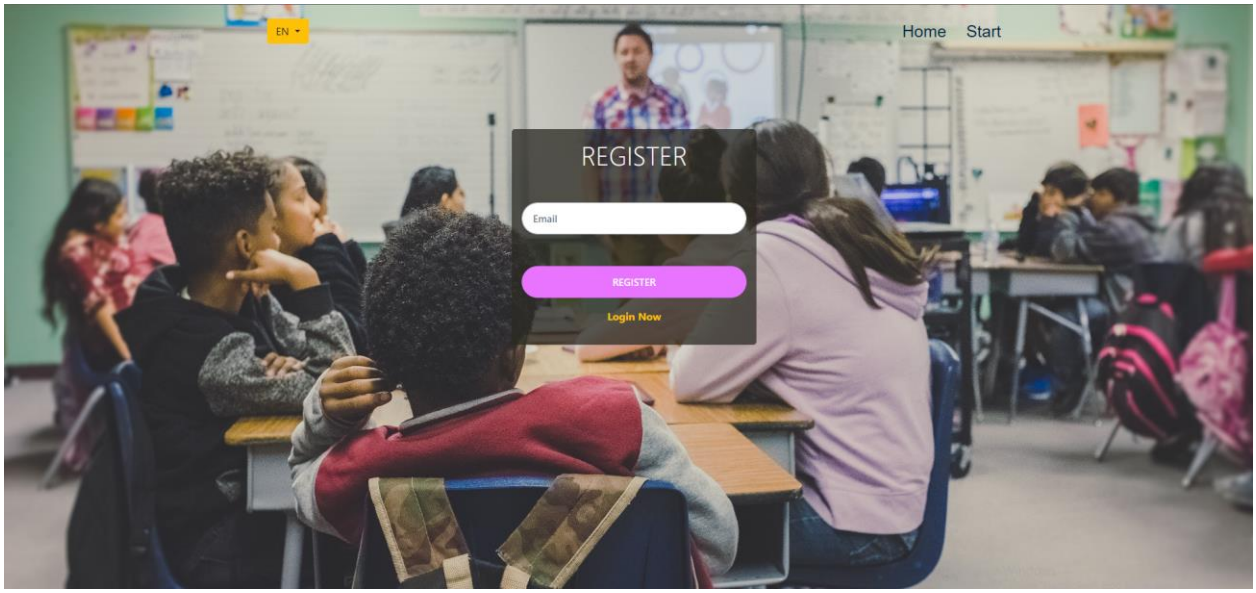

Рис. 4.4. Сторінка реєстрації

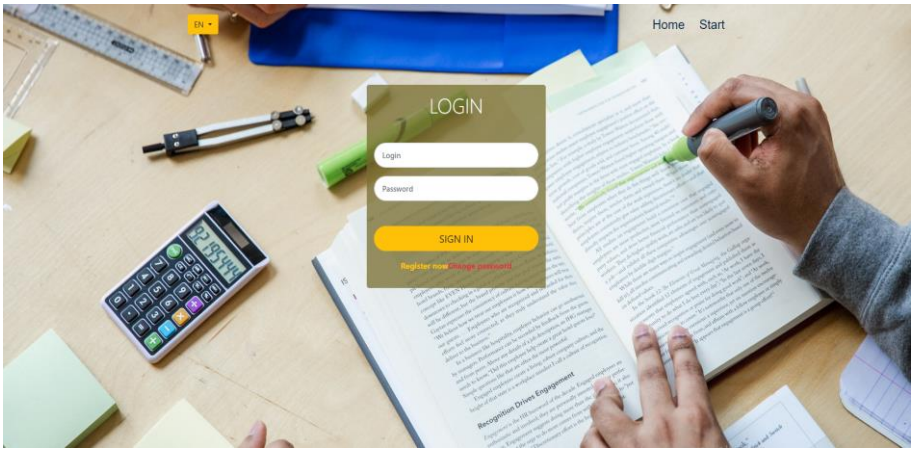

Рис. 4.5. Форма авторизації користувача

Після успішної авторизації користувач потрапляє на дешборд, де може вносити дані згідно, чат для спілкування з адміністрацією при поданні заявки, отримувати сповіщення.

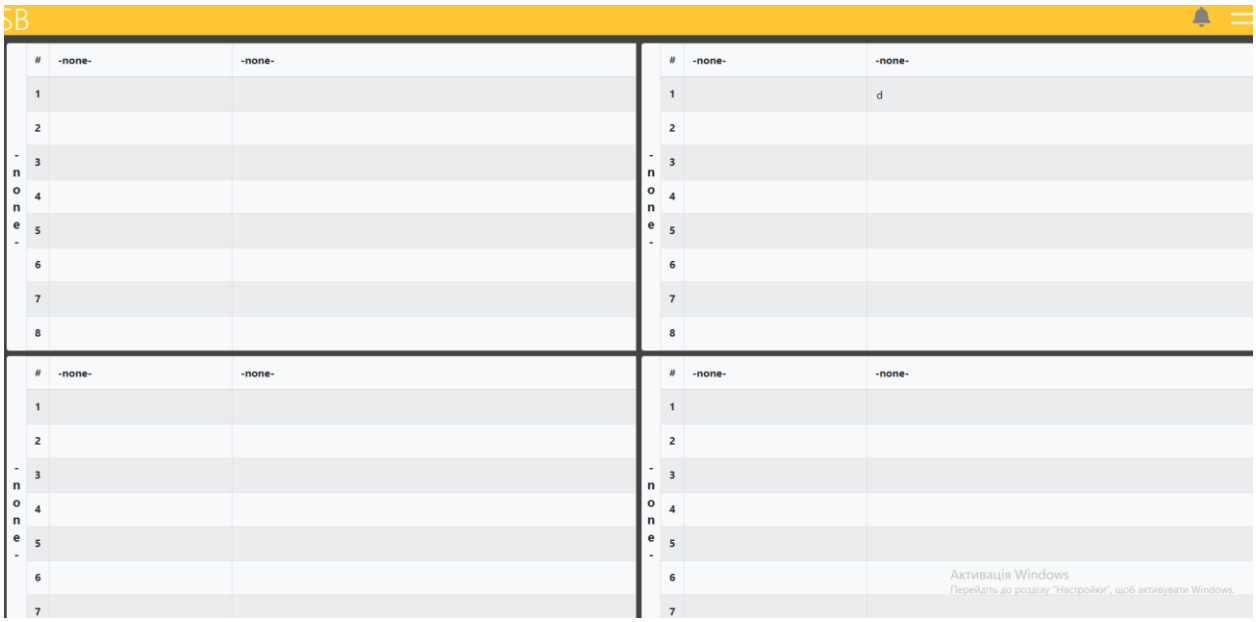

Рис. 4.6. Дашборд користувача

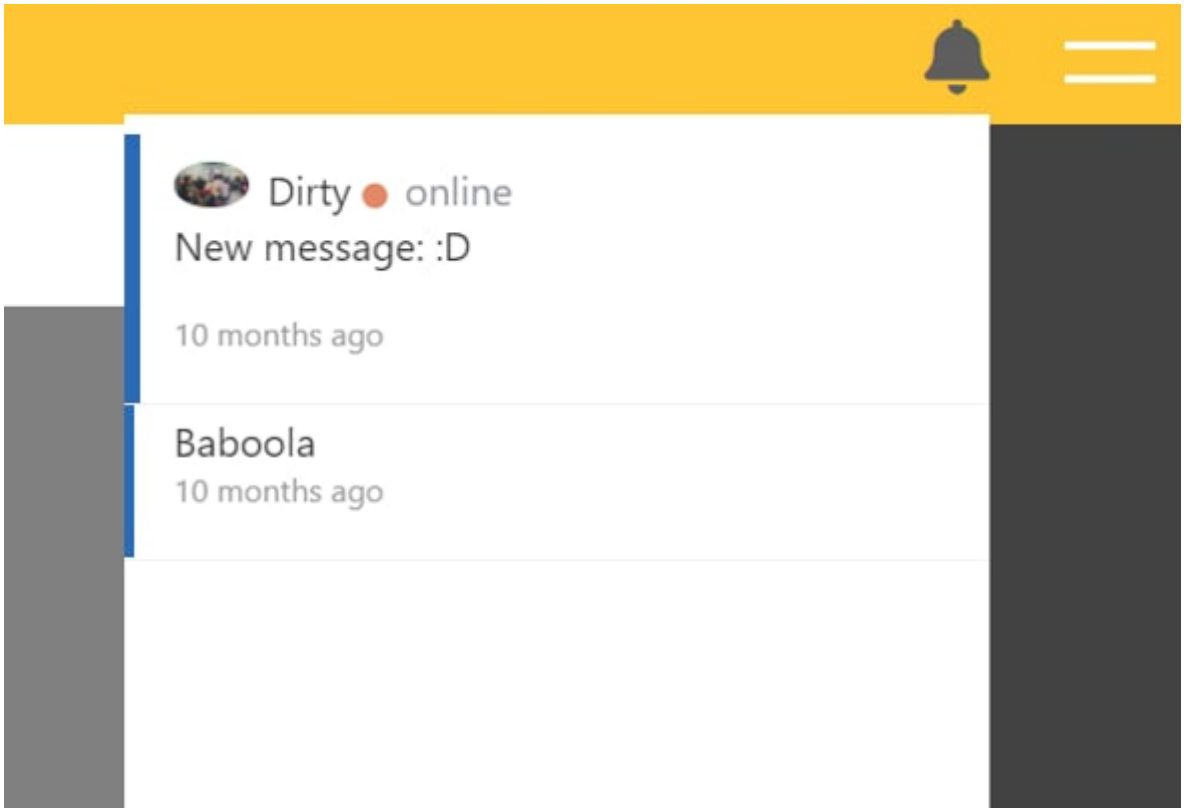

Рис. 4.7. Сповіщення про нове повідомлення

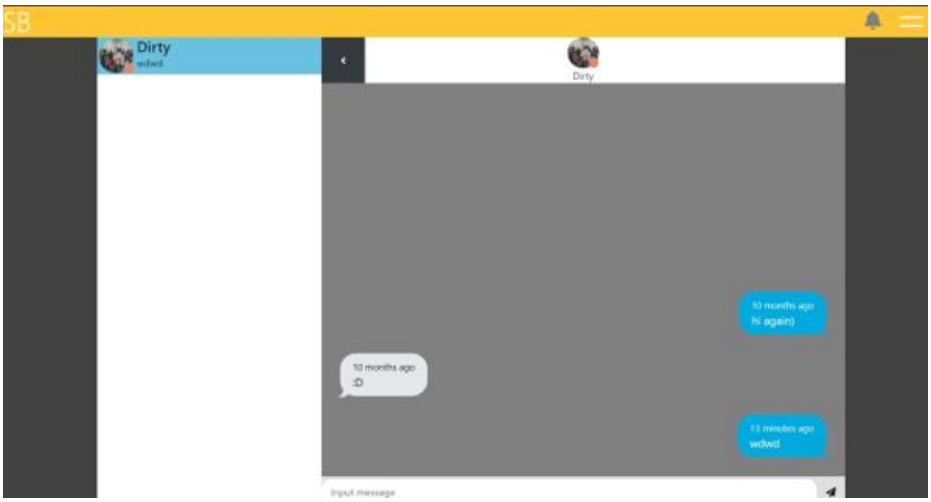

Рис. 4.8. Чат

З панелі адміністратора можна переглядати профілі користувачів, статусу та блокувати їх акаунти.

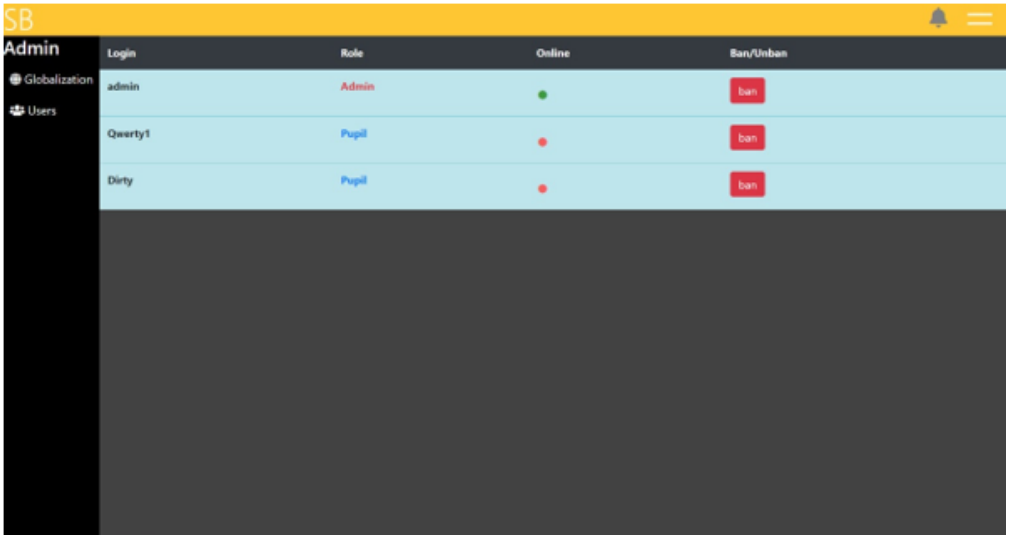

Рис. 4.9. Адмін панель з користувачами

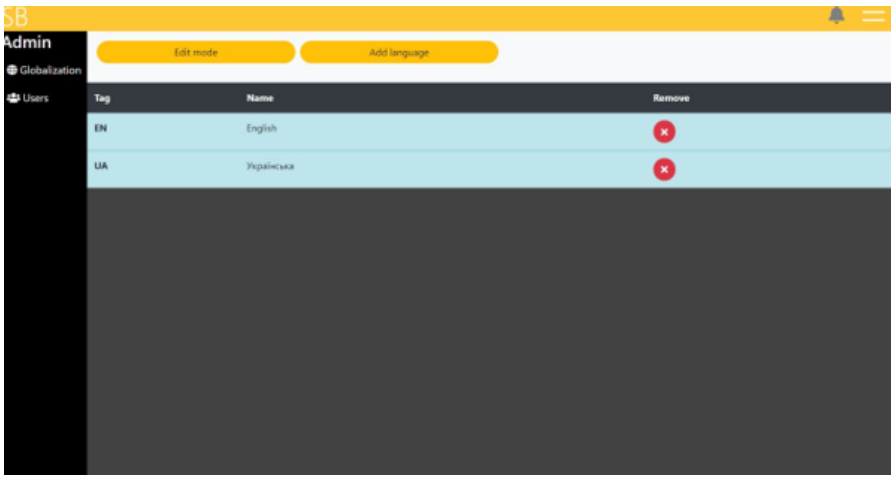

Рис. 4.10. Панель адміністрації для корегування та змін мов

Для того щоб додати мову, необхідно клацнути по кнопці “Add language” та прописати назву нової мови, та її коротка версія. Приклад, Українська (UA).

Далі необхідно вибрати мову сайту, яку потрібно прокорегувати, та нажати на кнопку “Edit language”. Після цього зверху появиться синня смужка з кнопками для виходу зі стану корегування, та перезавантаження моду без перегружання сторінки.

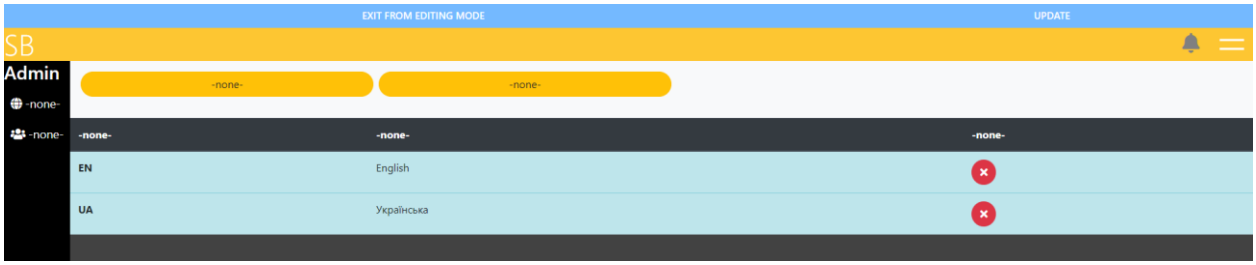

Рис. 4.11. Стан редагування мови

Тепер клацнувши по любому напису “-NONE-” з’явиться наступний інтерфейс для коректування:

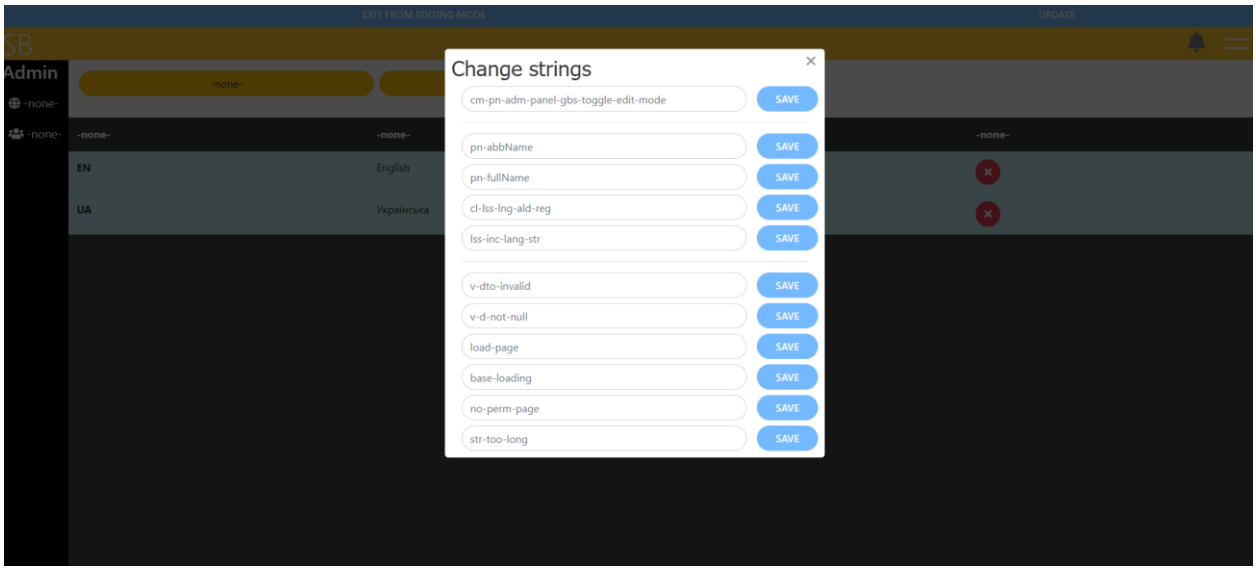

Рис. 4.12. Приклад зміни напису

Окрім зміни самого напису під потрібну мову, також є варіанти зміни локалізації можливих помилок, які можуть виникнути під час перебування на редагуємій сторінці.

Після проставлення значення натискаємо на кнопку “Save” та закриваємо вікно. Тепер можемо побачити що напис поля змінився.

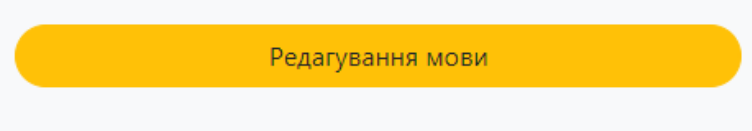

Рис. 4.12. Змінена кнопка.

Якщо ви змініте мову назад на англійську, то без перезавантаження сторінки відбудеться підміна всіх написав на вибрану локалізацію. Приклад наведений нище.

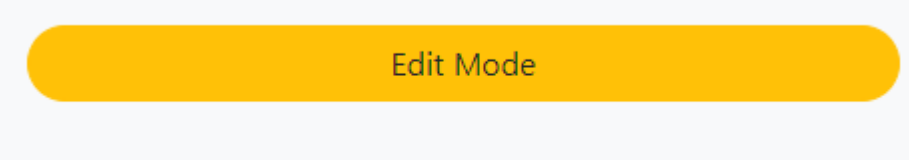

Рис. 4.13. Та сама кнопка після зміни мови

Не потрібно також забувати про “Settings”, де є можливість зміни логіну, аватарки. На даному етапі, на жаль, в цій сторінці тільки нереалізована функція загружання свого резюме на сайт.

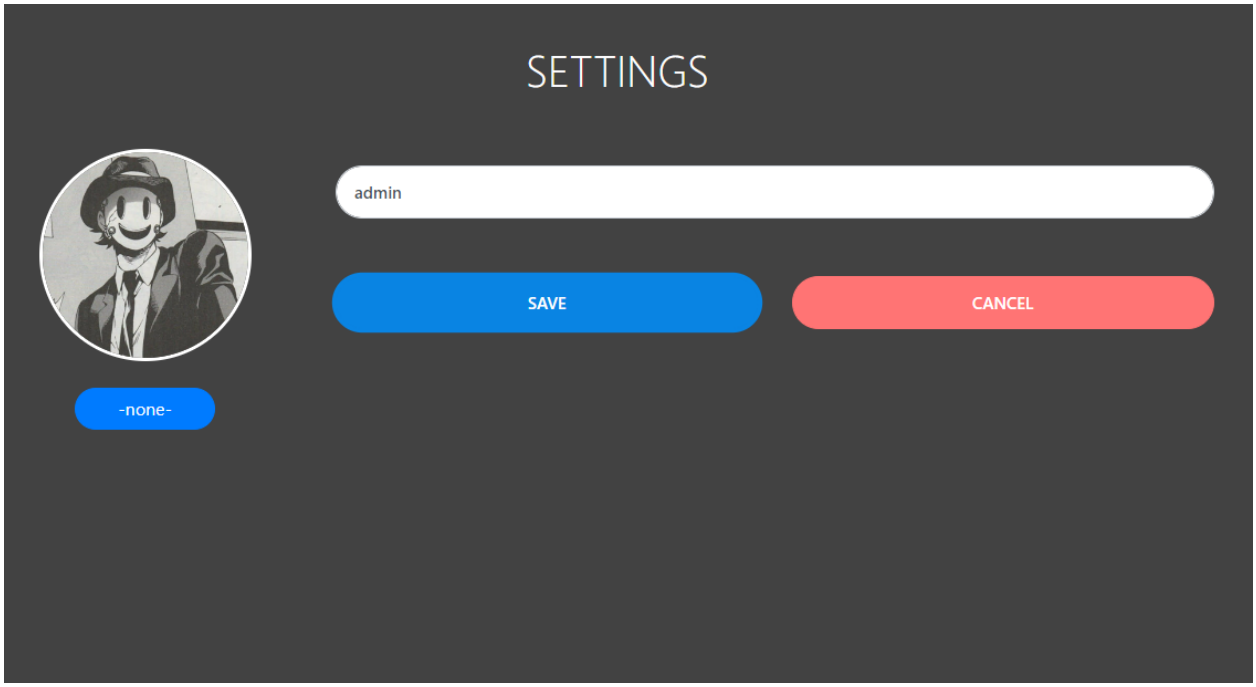

Рис. 4.14. Сторінка з налаштуваннями профілю

У майоутньому планується додати на цю сторінку кнопку для загружання CV на вебзастосунок, та функціоналу зміни пароля для користувача.

До речі, варто згадати також про те, що між сторінками зручно переходити через реалізовану ванель, яка може закриватись щоб не мішати авторизованим користувачам.

Висновок

У даній дипломній роботі був проведений аналіз існуючих вебдодатків для "Відділу кадрів" з метою виявлення їх проблем та недоліків. Цей аналіз дав змогу визначити певні проблемні аспекти, такі як обмежена функціональність, недостатня швидкодія, складність у використанні та інші.

Для вирішення цих проблем було запропоновано розроблення нового вебдодатку на базі технологій Angular та .NET Core. Використання цих технологій дозволить створити сучасний, ефективний та гнучкий вебдодаток з покращеними можливостями. Angular надає можливість створення зручного інтерфейсу користувача з багатофункціональними можливостями, тоді як .NET Core забезпечує високу продуктивність та кросс-платформену підтримку.

Розробка нового вебдодатку на основі вищезгаданих технологій дозволить покращити роботу "Відділу кадрів" шляхом автоматизації процесів, поліпшення доступності та зручності використання. Сучасні підходи та методи програмування технічних засобів та застосунків включає в себе багато аспектів: планування проекту, проектування архітектури додатку, розробка гнучких методів для оптимізації роботи застосунку, тестування різних версій додатку та його функціоналу, запуск проекту в продакшн, постійна його підтримка та допрацювання. Розвиток різноманітних програмних застосунків, фреймворків, бібліотек та обраної мови програмування допомагають розробнику зручніше створювати програмні продукти. прямолінійний та стрімкий розвиток сфери веб - програмування та її підтримки допомагає розробникам створювати необхідні ресурси, розвивати бізнес, спрощувати рутинні речі звичайних громадян.

Список використаної літератури

1. Angular URL: https://angular.io [1]
2. Angular Routing URL: https://angular.io/guide/routing-overview [2]
3. Karma js URL: https://karma-runner.github.io/latest/index.html [3]
4. Mark Thompson: Angular Projects: Build modern web apps by exploring Angular 12 with 10 different projects and cutting-edge technologies, 2nd Edition 2021p., 344c. [4]
5. Yakov Fain and Anton Moiseev: Angular Development with TypeScript, Second Edition 2018p., 560c. [5]

Додатки

Додаток А. Код серверу

```
namespace SchoolBridge.API
{

    public class Startup
    {
        private readonly IConfiguration _configuration;
        private readonly IWebHostEnvironment _env;

        //private readonly IEnumerable<string> _allowedOrgins = new string[] {
"http://192.168.0.4:4200", "http://192.168.0.4:4300" };

        public Startup(IConfiguration configuration, IWebHostEnvironment env)
        {
            _configuration = configuration;
            _env = env;
        }

        // This method gets called by the runtime. Use this method to add services
to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            // Initialization

            services.AddHostedService<ServiceInitializationHosting>();

            // Db Context
            services.AddDbContext<DbContext, ApplicationContext>(opt =>

opt.UseNpgsql(_configuration.GetSection("ConnectionStrings")["mypostgres"]),
optionsLifetime: ServiceLifetime.Singleton);

            // Mapper
            services.AddSingleton(new MapperConfiguration(mc =>
            {
                mc.AddProfile(new DomainMapperProfile());
            }).CreateMapper());

            services.AddSingleton<IJsonConverterService, JsonConverterService>();

            // Program status

            services.UseProgramStatusService(new ProgramStatusServiceConfiguration
            {
                CurrentStatusPath = "./Assets/ProgramStatus.json",
                DefaultStatus = new ProgramStatus
                {
                    IsLoadedFirst = true
                }
            });

            // Client errors
            services.UseClientErrorManager();

            // Signalr notification

            services.AddSingleton<IUserConnectionService, UserConnectionService>();
            services.AddSingleton<INotificationService, NotificationService>();


services.UsePermanentConnectionService(_configuration.GetSection("PermanentConnectio
nService"));
```

```
            services.UseOnlineService(_configuration.GetSection("OnlineService"));

services.UseChatEventService(_configuration.GetSection("ChatEventService"));
            services.AddScoped<IDataBaseNotificationService,
DataBaseNotificationService>();

            // File service
            services.UseFileService(new FileServiceConfiguration
            {
                SaveDirectory = _env.ContentRootPath + "/Assets/Files"
            });

            // Image Service
            services.UseImageService(new ImageServiceConfiguration());

            // Email Service
            services.UseEmailService(_configuration.GetSection("EmailService"));

            // Custom jwt auth
            services.UseJwtTokenService(_configuration.GetSection("JwtSettings"));

            // Scoped Services
            services.AddScoped(typeof(IGenericRepository<>),
typeof(GenericRepository<>));
            services.AddScoped<ScopedHttpContext>();
            services.AddScoped<IValidatingService, ValidatingService>();

            services.AddScoped<ILanguageStringService, LanguageStringService>();
            services.AddScoped<IRoleService, RoleService>();
            services.UseUserService(_configuration.GetSection("UserService"));
            services.AddScoped<IPermissionService, PermissionService>();
            services.AddScoped<IRegistrationService, RegistrationService>();
            services.AddScoped<ILoginService, LoginService>();
            services.AddScoped<IDirectMessagesService, DirectMessageService>();

            services.UseSubjectService(new SubjectServiceConfiguration());

services.UseLoginRegistrationService(_configuration.GetSection("RegistrationService"
));
            services.UseLanguageStringService(new LanguageStringServiceConfiguration
{ DefaultLanguage = "en" });

            services.AddScoped<IBanUsserService, BanUserService>();
            //SignalR

services.AddSignalR().AddNewtonsoftJsonProtocol().AddHubOptions<ServerHub>(options
=>
            {
                options.EnableDetailedErrors = true;
            });
            services.AddControllers();
        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            app.UseCors(x => x.SetIsOriginAllowed(_ => true/*(x) => {
Console.WriteLine(x); return _allowedOrgins.Contains(x); }*/)
                              .AllowAnyHeader()
                              .WithMethods("GET", "POST")
                              .AllowCredentials());

            app.UseDeveloperExceptionPage();
            app.UseMiddleware<ScopedHttpContextMiddleware>();
            app.UseMiddleware<ClientErrorsMiddleware>();
```

```
            app.UseRouting();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
                endpoints.MapHub<ServerHub>("/api/notify");
            });
        }
    }
}
```

Додаток Б. Приклад коду контроллера

```
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using SchoolBridge.API.Controllers.Attributes;
using SchoolBridge.API.Controllers.Attributes.Globalization;
using SchoolBridge.API.Controllers.Attributes.Validation;
using SchoolBridge.Domain.Services.Abstraction;
using SchoolBridge.Helpers.DtoModels;
using SchoolBridge.Helpers.DtoModels.Globalization;

namespace SchoolBridge.API.Controllers
{
    [Route("api/[controller]/[action]")]
    public class GlobalizationController : ControllerBase
    {
        private readonly ILanguageStringService _languageStringService;
        public GlobalizationController(ILanguageStringService languageStringService)
        {
            _languageStringService = languageStringService;
        }

        [HttpPost, ActionName("language/add"), HasPermission("CreateLanguage")]
        public async Task<ResultDto> AddLanguage([FromBody,
MyValidation]AddLanguageDto entity)
        {
            _languageStringService.AddLanguage(entity.AbbName, entity.FullName);
            return ResultDto.Create(entity);
        }

        [HttpGet, ActionName("language/remove"), HasPermission("RemoveLanguage")]
        public async Task<ResultDto> RemoveLanguage([ArgValid("str-input", "lss-lng-
name")]string abbName)
        {
            _languageStringService.RemoveLanguage(abbName);
            return ResultDto.Create(null);
        }

        [HttpPost, ActionName("language/edit"), HasPermission("EditLanguage")]
        public async Task<ResultDto> EditLanguage([FromBody,
MyValidation]GetLanguageEditDto entity)
        {
            _languageStringService.EditLanguage(entity.AbbName, entity.FullName);
            return ResultDto.Create(null);
        }

        [HttpPost, BaseUpatedId]
        public async Task<ResultDto> Strings([FromBody,
MyValidation]GetLanguageStringsDto entity)
        {
            return
ResultDto.Create(_languageStringService.GetByStringNameCurrent(entity.Strings));
```

```
        }

        [HttpPost, BaseUpatedId]
        public async Task<ResultDto> StringsByType([FromBody,
MyValidation]GetLanguageStringsDto entity)
        {
            return
ResultDto.Create(_languageStringService.GetByTypeCurrent(entity.Strings));
        }

        [HttpGet]
        public async Task<ResultDto> Info()
        {
            return ResultDto.Create(_languageStringService.GetGlobalizationInfo());
        }

        [HttpPost, ActionName("string/addorupdate"), HasPermission("EditGbStrings")]
        public async Task<ResultDto> StringAddOrUpdate([FromBody,
MyValidation]AddOrUpdateStringDto entity)
        {
            _languageStringService.AddOrUpdateString(entity.Name,
_languageStringService.GetCurrentLanguage(), entity.String);
            return ResultDto.Create(null);
        }

        [HttpGet, ActionName("update-baseupdateid"),
HasPermission("UpdateBaseUpdateId")]
        public async Task<ResultDto> UpdateBaseUpdateId()
        {
            _languageStringService.UpdateBaseUpdateId();
            return ResultDto.Create(_languageStringService.GetGlobalizationInfo());
        }
    }
}
```

Додаток С. Приклад коду сервіса

```
using SchoolBridge.Domain.Services.Abstraction;
using SchoolBridge.Domain.Services.Configuration;
using SchoolBridge.Helpers.AddtionalClases;
using SchoolBridge.Helpers.AddtionalClases.EmailService;
using SchoolBridge.Helpers.Extentions;
using SchoolBridge.Domain.Managers.CClientErrorManager;
using System;
using System.Collections.Generic;
using System.Net.Mail;
using System.Text;


namespace SchoolBridge.Domain.Services.Implementation
{
    public class EmailService : IEmailService
    {
        private readonly EmailServiceConfiguration _configuration;
        private readonly PriorityQueue<EmailEntity> _queue = new
PriorityQueue<EmailEntity>();
        public PriorityQueue<EmailEntity> EmailQueue { get => _queue; }

        public EmailService(EmailServiceConfiguration configuration) {
            _configuration = configuration;
        }

        public static void OnInit(ClientErrorManager manager) {
```

```
            manager.AddErrors(new ClientErrors("EmailService", new
Dictionary<string, ClientError>
            {

            }));
        }
        private string CreateDraftPath(string name) {
            return _configuration.DraftsPath + "/" + name + ".draft.html";
        }
        private string GetDraft(string name) {
            return System.IO.File.ReadAllText(CreateDraftPath(name));
        }

        private string ComposeDraftBody(string draftBody, params string[] arguments)
        {
            foreach (var item in arguments)
                draftBody = draftBody.ReplaceFirstOccurrance("$arg$", item);
            return draftBody;
        }

        private string ComposeDraft(string draft, params string[] arguments)
        {
            return ComposeDraftBody(GetDraft(draft), arguments);
        }

        public void Send(string toEmail, string FromEmail, string subject, string
body, SendCompleatedEventHandler eventHandler, EmailEntityPriority priority, object
AddtionalInfo)
        {
            MailMessage oMailMsg = new MailMessage();
            oMailMsg.To.Add(toEmail);
            oMailMsg.Subject = subject;
            oMailMsg.From = new MailAddress(FromEmail, FromEmail, Encoding.UTF8);

            oMailMsg.IsBodyHtml = true;
            oMailMsg.Body = body;

            _queue.Enqueue(new EmailEntity
            {
                Message = oMailMsg,
                StartSendingTime = DateTime.Now,
                CompletedEventHandler = eventHandler,
                AddtionalInfo = AddtionalInfo
            }, (int)priority);
        }

        public void SendByDraft(string toEmail, string FromEmail, string subject,
string draft, SendCompleatedEventHandler eventHandler, EmailEntityPriority priority,
object AddtionalInfo, params string[] arguments)
        {
            Send(toEmail, FromEmail, subject, ComposeDraft(draft, arguments),
eventHandler, priority, AddtionalInfo);
        }

        public void SendDefault(string toEmail, string subject, string body,
SendCompleatedEventHandler eventHandler, EmailEntityPriority priority, object
AddtionalInfo)
        {
            Send(toEmail, _configuration.DefaultSendFrom, subject, body,
eventHandler, priority, AddtionalInfo);
        }

        public void SendDefaultByDraft(string toEmail, string subject, string draft,
SendCompleatedEventHandler eventHandler, EmailEntityPriority priority, object
AddtionalInfo, params string[] arguments)
```

```
        {
            Send(toEmail, _configuration.DefaultSendFrom, subject,
ComposeDraft(draft, arguments), eventHandler, priority, AddtionalInfo);
        }
    }
}
```

Додаток Г. Приклад коду клієнтської частини

```
import {
  Subject,
  BehaviorSubject,
  Observable,
  concat,
  of,
  forkJoin,
} from "rxjs";
import { MyLocalStorageService } from "src/app/Services/my-local-
storage.service";
import {
  tap,
  bufferWhen,
  mergeMap,
  debounceTime,
  map,
  concatAll,
  bufferCount, throttleTime, throttle, delayWhen, finalize
} from "rxjs/operators";
import { Injectable } from "@angular/core";
import { HttpGlobalizationService } from "./http-globalization.service";
import { GlobalizationInfoService } from "./globalization-info.service";
import { IsLoading } from 'src/app/Helpers/is-loading.class';

class GbString {
  private static readonly regex = /\[([a-z-0-9]+)(\s?\,\s?.+)?\]/;

  public readonly baseObs: Observable<string>;
  public readonly id: string;
  public readonly args: (string | GbString)[] = [];
  public readonly concatedObs: Observable<string>;

  constructor(
    private readonly _gbsService: GlobalizationStringService,
    private readonly _parent: GbString,
    str: string
  ) {
    const match = str.match(GbString.regex);
    if (!match || match.length === 0) throw "Error string " + str;
    this.id = match[1];
    this.baseObs = _gbsService.getStringObs(this.id);

    if (match[2])
      match[2].split(/\s?\,\s?/).forEach((r) => {
```

```
        if (!r || r.length === 0) return;
        if (!r.match(GbString.regex)) this.args.push(r);
        else this.args.push(new GbString(_gbsService, this, r));
      });
    if (!_parent)
      this.concatedObs = this.baseObs.pipe(
        map((x) => this.createRootString(x))
      );
    //else this.baseObs.subscribe(x => this.update());
  }

  public createString(strs: string[]): string {
    //console.log(strs);
    let som = strs[0];
    strs.forEach((x, i) => {
      if (i === 0) return;
      som = som.replace("$arg$", x);
    });
    return som;
  }

  public createRootString(root: string): string {
    return this.createString([
      root,
      ...this.args.map((x) => (typeof x === "string" ? x : x.createMyString())),
    ]);
  }

  public createMyString(): string {
    return this.createRootString(
      this._gbsService.getLoadedStringSave(this.id, "-none-")
    );
  }

  public update(){
    if (this._parent)
      this._parent.update();
    else this._gbsService.updateString(this.id);
  }
}

@Injectable()
export class GlobalizationStringService {
  private _stringDATA: Record<string, Record<string, string>> = {};
  private get _stringD() {
    if (this._gbInfoService.info){
      if (this._stringDATA[this._gbInfoService.info.currentLanguage])
        return this._stringDATA[this._gbInfoService.info.currentLanguage];
      return this._stringDATA[this._gbInfoService.info.currentLanguage] = {};
    }
```

```
    return {};
  }
  private _toggleMainGetBuffer: Subject<any> = <Subject<any>>(
    new Subject<any>().pipe(
      debounceTime(10),
      delayWhen(x => this._gbInfoService.infoObs),
    )
  );
  private _mainGetThread: Subject<string> = null;

  private _initializedStrings: Record<string, BehaviorSubject<string>> = {};

  public readonly loading: IsLoading = new IsLoading();

  constructor(
    private _localStorage: MyLocalStorageService,
    private _httpGbService: HttpGlobalizationService,
    private _gbInfoService: GlobalizationInfoService
  ) {
    if (this._localStorage.isIssetKey("gbdata"))
      this._stringDATA = this._localStorage.read("gbdata");

    this._gbInfoService.infoObs.subscribe((x) => {
      if (x) {
        if (!Object.keys(this._stringDATA).includes(x.currentLanguage))
          this._stringDATA[x.currentLanguage] = {};
      }
    });

    this._mainGetThread = <Subject<string>>new Subject<string>().pipe(
      tap((x) => this._toggleMainGetBuffer.next(null)),
      bufferWhen(() => this._toggleMainGetBuffer),
      mergeMap((x) => {
        x.forEach((x) => {
          if (this._initializedStrings[x])
            this._initializedStrings[x].next("-loading-");
        });
        this.loading.status = true;
        return this._httpGbService.getStrings(x).pipe(
          map((r) => {
            Object.assign(this._stringD, r);
            this._localStorage.write("gbdata", this._stringDATA);
            x.forEach((m) => {
              if (!Object.keys(r).includes(m))
                r[m] = this.getLoadedStringSave(m, "-none-");
            });
            Object.keys(r).forEach((m) => {
              if (
                Object.keys(this._initializedStrings).includes(m) &&
                this._initializedStrings[m]
```

```
            )
              this._initializedStrings[m].next(r[m]);
          });
          return "";
        }),
        finalize(() => this.loading.status = false)
      );
    }),

  );

  this._mainGetThread.subscribe();
}

public loadAllNoLoadedString() {
  Object.keys(this._initializedStrings).forEach((r) => {
    if (!Object.keys(this._stringD).includes(r)) this._mainGetThread.next(r);
    else if (this._initializedStrings[r])
      this._initializedStrings[r].next(this._stringD[r]);
  });
}

public updateString(r: string){
  this._initializedStrings[r].next(this._stringD[r]);
}

public getLoadedStringSave(str: string, noFund: string): string {
  return Object.keys(this._stringD).includes(str)
    ? this._stringD[str]
    : noFund;
}

private isInitializedString(str: string) {
  return Object.keys(this._initializedStrings).includes(str);
}

private isLoadedString(str: string) {
  return Object.keys(this._stringD).includes(str);
}

public getLoadedStringHttp(str: string, noFund: string): string {
  if (Object.keys(this._stringD).includes(str))
    return this._stringD[str];
  else {
    this._mainGetThread.next(str);
    return noFund;
  }
}

public initConstStrings(str: string[]) {
```

```
  str.forEach((x) => {
    if (!this.isInitializedString(x))
      this._initializedStrings[x] = null;
    this.getLoadedStringHttp(x, "-loading-");
  });
}

public getStringsObs(str: string[]): Record<string, Observable<string>> {
  const obss: Record<string, Observable<string>> = {};
  str.forEach((x) => (obss[x] = this.getStringObs(x)));
  return obss;
}

public getStringObs(str: string): Observable<string> {
  if (
    !this.isInitializedString(str) ||
    this._initializedStrings[str] == null
  ) {
    this._initializedStrings[str] = new BehaviorSubject<string>(
      this.getLoadedStringHttp(str, "-loading-")
    );
  }
  return this._initializedStrings[str];
}

public convertString(str: string): Observable<string> {
  return new GbString(this, null, str).concatedObs;
}

public getConstStringName(str: string) {
  const regex = /\[([a-z-0-9]+)(\s?\,\s?.+)?\]/;
  const match = str.match(regex);
  if (!match || match.length === 0) throw "Error string " + str;
  return match[1];
}

public setString(name: string, str: string) {
  this._stringD[name] = str;
  this._localStorage.write("gbdata", this._stringDATA);
  if (
    Object.keys(this._initializedStrings).includes(name) &&
    this._initializedStrings[name]
  )
    this._initializedStrings[name].next(str);
}

public clearAllData() {
  this._stringDATA = {};
  this._localStorage.write("gbdata", this._stringDATA);
}
```