Державний торговельно-економічний університет
Кафедра інженерії програмногозабезпечення та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Професійно-орієнтована експертна система підбору програмного забезпечення за запитами ІТ-фахівців»

Студента 2м курсу, 2 групи, спеціальності 121 «Інженерія програмного забезпечення» освітньої програми «Інженерія програмного забезпечення» _____ підпис студента — Осадчука Миколи Ярославовича

Науковий керівник кандидат економічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Тищенко Дмитро Олександрович

Гарант освітньої програми кандидат педагогічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис гаранта — Котенко Наталія Олексіївна

КИЇВ – 2023

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Освітня програма 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«13» грудня 2022 р.

Завдання
на випускну кваліфікаційну роботу студентові

Осадчуку Миколі Ярославовичу
(прізвище, ім'я, по батькові)

1. «Професійно-орієнтована експертна система підбору програмного забезпечення за запитами ІТ-фахівців»

Затверджена наказом ректора від «06» грудня 2022 р. №3285

2. Строк здачі студентом закінченої роботи 27 листопада 2023

3. Цільова установка та вихідні дані до роботи

Мета роботи - розробка та валідація професійно-орієнтованої експертної системи, яка дозволить ІТ-фахівцям ефективно вибирати програмне забезпечення на основі їх специфічних запитів та потреб

Об'єкт дослідження – **:** Процес підбору та вибору програмного забезпечення ІТ-фахівцями.

Предмет дослідження - Експертна система, що використовує алгоритми ШІ для аналізу запитів ІТ-фахівців та визначення найбільш підходящого програмного забезпечення з урахуванням вказаних параметрів користувача.

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1 АНАЛІЗ ДОКУМЕНТАЦІЇ ТА ДОСЛІДЖЕНЬ ВЗАЄМОДІЇ КОРИСТУВАЧА З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

1.1. Дослідження потреб користувача

1.2. Важливість дослідження великих, відомих організацій і платформ

1.3. Перелік та опис ключових організацій ІТ галузі

1.4. Стандарти якості ПЗ

1.5. Міжнародні документи норм програмного забезпечення

1.6 I.SO/IEC 25010

1.7. Існуючі методи підбору ПЗ, їх переваги та недоліки

1.8. Висновки до розділу 1

РОЗДІЛ 2 ІНСТРУМЕНТИ РОЗРОБКИ ПРОЕКТУ

2.1. Важливість вибору інструметів

2.2. «IDE» проекту

2.3. MySQL Workbench

2.4. DevTools

2.5. Front-end

2.7. Back-end

2.8. База даних

2.9. Поштовий сервіс

2.10. Google OAuth

2.10. Хмарні сервіси AWS

2.11. User flow

6. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*	*07.11.2022*	*07.11.2022*
2.	*Розробка та затвердження завдання на роботу магістра (стац/заоч)*	*13.12.2022*	*13.12.2022*
3.	*Вступ та перелік літературних джерел*	*24.02.2023*	*24.02.2023*
4.	*Розробка технічного завдання*	*15.03.2023*	*15.03.2023*
5.	*Розділ 1 Аналіз документації та досліджень взаємодії користувача з програмним забезпеченням*	*10.04.2023*	*10.04.2023*
6.	*Розділ 2 Інструменти розробки проекту*	*24.05.2023*	*24.05.2023*
7.	*Розділ 3 Інформаційне забезпечення експертної системи*	*06.09.2023*	*06.09.2023*
8.	*Розділ 4 Опис розробки програмного забезпечення*	*01.10.2023*	*01.10.2023*
9.	*Розробка програми та методики тестування*	*18.10.2023*	*18.10.2023*
10.	*Написання наукової статті*	*17.05.2023*	*17.05.2023*
11.	*Керівництво користувача*	*25.10.2023*	*25.10.2023*
12.	*Висновки та пропозиції*	*01.11.2023*	*01.11.2023*
13.	*Здача випускної кваліфікаційної роботи на кафедру (перша перевірка)*	*06.11.2023*	*06.11.2023*
14.	*Підготовка автореферату та презентації доповіді*	*06.11.2023*	*06.11.2023*
15.	*Попередній захист випускної кваліфікаційної роботи*	*20.11.2023 – 24.11.2023*	*20.11.2023 – 24.11.2023*
16.	*Здача зброшурованої випускної кваліфікаційної роботи*	*27.11.2023*	*27.11.2023*
17.	*Зовнішнє рецензування випускної кваліфікаційної роботи*	*29.11.2023*	*29.11.2023*
18.	*Підготовка до публічного захисту випускної кваліфікаційної роботи*	*05.12.2023- 06.12.2023*	*05.12.2023- 06.12.2023*

7. Дата видачі завдання «13» грудня 2022 р.

8. Науковий керівник випускної кваліфікаційної роботи Тищенко Д.О.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Котенко Н.О.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Осадчук М.Я.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проєкту

Науковий керівник випускної кваліфікаційної роботи

_____

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента _____Осадчук М.Я._____

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____Котенко Н.О._____

(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О. В._____

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Ця робота присвячена розробці та аналізу професійно-орієнтованої експертної системи, призначеної для автоматизації процесу підбору програмного забезпечення за специфічними запитами IT-спеціалістів. З огляду на швидкий розвиток IT-галузі та постійне оновлення спектру програмних продуктів, існує значна потреба в оптимізації вибору програмного забезпечення, що відповідає конкретним професійним вимогам. Робота включає в себе аналіз існуючих рішень, визначення критеріїв для оцінки програмного забезпечення, розробку алгоритмічної бази експертної системи та реалізацію прототипу системи. Описано проектування та розробка експертної системи. Створено зручний користувацький інтерфейс.

Ключові слова: розробка, вибір програмного забезпечення, IT-галузь.

ABSTRACT

This paper is devoted to the development and analysis of a professionally oriented expert system designed to automate the process of software selection according to specific requests of IT specialists. Given the rapid development of the IT industry and the constant updating of the range of software products, there is a significant need to optimise the selection of software that meets specific professional requirements. The work includes analysing existing solutions, defining criteria for software evaluation, developing an algorithmic framework for an expert system, and implementing a prototype system. The design and development of the expert system is described. A user-friendly user interface has been created.

Keywords: development, software selection, IT industry.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних.

БекЕнд, Бек – серверна частина програми

ФронтЕнд, фронт– користувацька частина програми

ЗМІСТ

					ДТЕУ 121 02-13.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		01.11.23	*Професійно-орієнтована експертна система підбору програмного забезпечення за запитами ІТ-фахівців*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Тищенко Д.О.		01.11.23		*Зміст*	*3*	*62*
Гарант		Котенко Н.О.		01.11.23		*Факультет інформаційних технологій 2м курс, 2 група*		
Розробив		Осадчук М.Я.		01.11.23	*Зміст*			

ВСТУП

Актуальність теми

В епоху цифрової трансформації, сфера ІТ стає все більш значущою і невід'ємною частиною сучасного суспільства. Швидкий розвиток технологій, постійне зростання кількості програмного забезпечення викликає необхідність у високоякісних інструментах для підбору оптимальних програмних продуктів, що відповідали б індивідуальним потребам ІТ-фахівців.

Актуальність обраної теми обумовлена цими чинниками, а також необхідністю забезпечення ефективної взаємодії між розробниками, кінцевими користувачами та іншими учасниками ІТ-процесів.

Правильний підбір програмного забезпечення (ПЗ) в сучасному ІТ-світі набуває особливої актуальності з кількох причин:

Економія ресурсів: Придбання, розгортання та підтримка ПЗ може коштувати компаніям значних ресурсів. Правильний підбір дозволяє компаніям максимізувати свої інвестиції, запобігаючи надмірним витратам на непотрібне або непридатне програмне забезпечення.

Продуктивність: Ефективно підібране ПЗ може значно підвищити продуктивність роботи. З іншого боку, ПЗ, яке не відповідає потребам користувача, може стати причиною затримок та непродуктивних робочих процесів.

Сумісність: Велике різноманіття ПЗ доступне на ринку, і не всі програми сумісні між собою. Неправильний підбір може призвести до конфліктів між різними додатками та системами.

Скорочення часу на навчання: Правильно підібране ПЗ може бути більш інтуїтивним і простим у використанні, що зменшує час, потрібний для навчання співробітників.

Інтеграція з іншими системами: В сучасних компаніях часто використовується багато різних ІТ-систем. ПЗ, яке легко інтегрується з іншими додатками та сервісами, може значно спростити автоматизацію робочих процесів.

Адаптивність до змін: ІТ-галузь постійно розвивається, тому ПЗ повинно бути гнучким та здатним адаптуватися до змін у бізнес-процесах або технологіях.

Отже, в сучасному ІТ-світі правильний підбір ПЗ не просто бажаний, але і критично важливий для успіху, стабільності та безпеки будь-якої організації.

Об'єктом дослідження є процес вибору і рекомендації програмного забезпечення відповідно до специфікацій та потреб ІТ-спеціалістів.

Предметом дослідження є методи та засоби, які використовуються в експертних системах для автоматичного підбору програмного забезпечення.

Метою даного дослідження є розробка професійно-орієнтованої експертної системи, яка дозволить автоматизувати процес підбору програмного забезпечення за запитами ІТ-фахівців, враховуючи їх професійний досвід, потреби та інші важливі параметри.

Завдання проекту включають:

- Аналіз існуючих методів підбору програмного забезпечення.
- Визначення критеріїв та параметрів для ефективного підбору ПЗ.
- Розробка алгоритмів та методів для експертної системи.
- Тестування та валідація розробленої системи на реальних даних.

Наукова новизна дослідження полягає в комбінації сучасних методів машинного навчання, експертних систем та алгоритмів аналізу даних для

створення високоефективної системи підбору програмного забезпечення, яка буде максимально відповідати потребам IT-фахівців.

РОЗДІЛ 1
АНАЛІЗ ДОКУМЕНТАЦІЇ ТА ДОСЛІДЖЕНЬ ВЗАЄМОДІЇ КОРИСТУВАЧА З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

1.1. Дослідження потреб користувача

За минулих 5 років було проведено безліч досліджень в галузі IT, в тому числі і щодо потреб та вимог IT-спеціалістів до програмного забезпечення. Деякі з цих досліджень публікуються у провідних наукових журналах, конференціях або проводяться крупними аналітичними компаніями та дослідницькими інститутами.

Ось декілька загальних тенденцій та ключових вимог, які IT-спеціалісти шукали в ПЗ протягом останніх років:

Безпека

В зв'язку із зростаючим числом кіберзагроз, безпека даних стала на перший план. Основні вимоги стосуються шифрування даних, двофакторної автентифікації, політик доступу тощо.

Гнучкість і масштабованість

IT-спеціалісти шукають рішення, які легко інтегруються з іншими системами та можуть масштабуватися залежно від потреб організації.

Обчислювальні можливості в хмарі

З розвитком хмарних технологій, багато спеціалістів шукають рішення, які дозволяють проводити обчислення в хмарі.

Сумісність і інтеграція

Системи, які легко інтегруються з іншими інструментами та платформами, отримують перевагу.

Підтримка розробників

ІТ-спеціалісти шукають ПЗ з активною спільнотою, регулярними оновленнями та хорошою документацією.

Ліцензійна політика

ІТ-фахівці віддають перевагу програмному забезпеченню з гнучкими умовами ліцензування.

Зручний користувацький інтерфейс

Простота та інтуїтивність користувацького інтерфейсу також є важливим критерієм вибору.

Мобільність

З зростанням використання мобільних пристроїв, можливість користуватися ПЗ на різних пристроях стає все важливішою.

1.2. Важливість дослідження великих, відомих організацій і платформ

Дослідження великих, відомих організацій та платформ відіграють критичну роль в ІТ-сфері з багатьох причин. Ці організації мають доступ до великої кількості даних та ресурсів, що дозволяє їм аналізувати та ідентифікувати головні тенденції в індустрії.Великі організації часто розробляють та пропонують стандарти, які можуть стати загальноприйнятими у галузі, сприяючи її розвитку та інноваціям. Дослідження, проведені авторитетними інститутами чи компаніями, надають більше довіри і їх частіше використовують як джерело для прийняття рішень на різних рівнях.

Компанії та розробники можуть використовувати ці дослідження для прогнозування майбутніх потреб ринку та планування своєї стратегії розвитку. Результати таких досліджень часто стають темою для обговорення у ІТ-спільноті, сприяючи обміну думками, критичному мисленню та

інноваційному підходу. Дослідження допомагають навчальним закладам та освітнім програмам залишатися актуальними, адаптуючи свої курси та навчальні плани до змінюваних вимог ринку.Для керівників, інвесторів та інших зацікавлених сторін інформація з таких досліджень може слугувати ключем до прийняття важливих стратегічних рішень.

Через дослідження можна ідентифікувати потенційні загрози та вразливості у популярних технологіях, що допомагає підвищити рівень безпеки в індустрії. Дослідження великих, відомих організацій створюють основу для поглибленого розуміння стану IT-галузі, її тенденцій, можливостей та викликів.

1.3. Перелік та опис ключових організацій IT галузі

Stack Overflow Developer Survey: Щорічне дослідження, яке проводить Stack Overflow, збирає відгуки від розробників з усього світу щодо їх уподобань, використання технологій, потреб та вимог до інструментів.

GitHub's State of the Octoverse: Це щорічний звіт від GitHub, який аналізує активність на платформі і висвітлює тенденції в розробці ПЗ, в тому числі популярність мов програмування, бібліотек та інших інструментів.

Gartner's Magic Quadrant: Хоча це дослідження фокусується на оцінці постачальників програмного забезпечення в різних категоріях, воно також висвітлює ключові вимоги та потреби IT-організацій, на які звертають увагу при виборі рішень

Forrester Wave Reports: Аналогічно до Gartner's Magic Quadrant, Forrester Wave зосереджується на оцінці ПЗ у різних категоріях, але також аналізує потреби IT-фахівців.

World Quality Report by Capgemini: Цей звіт аналізує тенденції в області забезпечення якості та тестування ПЗ, включаючи потреби та вимоги IT-спеціалістів у цій сфері

1.4. Стандарти якості ПЗ

Стандарти якості програмного забезпечення (ПЗ) відіграють критично важливу роль у сучасній IT-індустрії. Вони створені з метою гарантувати, що програмне забезпечення відповідає визначеним критеріям надійності, продуктивності та інших ключових атрибутів.

Дотримання стандартів якості гарантує, що ПЗ буде надійним і стабільним у роботі. Це зменшує ризик втрати даних, збоїв системи та інших непередбачених проблем.Коли користувач або організація купують ПЗ, вони очікують, що воно працюватиме належним чином. Стандарти якості забезпечують, що продукт відповідає їхнім очікуванням. Виявлення та вирішення проблем на ранніх етапах розробки завдяки дотриманню стандартів якості може суттєво зменшити витрати на підтримку та виправлення помилок у майбутньому. Стандарти якості часто включають вимоги до сумісності, гарантуючи, що ПЗ може інтегруватися та працювати разом з іншими системами. Дотримання стандартів може привести до зменшення кількості помилок у коді, що, у свою чергу, приводить до швидшої розробки та меншої кількості виправлень. Стандарти якості можуть полегшити процес аудиту та перевірки ПЗ, адже вони надають чітке розуміння того, чого очікувати від програмного продукту. Організації, які дотримуються стандартів якості, можуть отримати конкурентні переваги, підвищуючи довіру до своїх продуктів серед споживачів та партнерів.

1.5. Міжнародні документи норм програмного забезпечення

Документи, що визначають критерії якості для програмного забезпечення, часто базуються на міжнародних стандартах і рекомендаціях. Ключові стандарти та документи будуть наведені нижче.

ISO/IEC 9126 (зараз відомий як ISO/IEC 25010): Цей стандарт описує модель якості ПЗ і включає такі характеристики як функціональна придатність, надійність, використання, ефективність, здатність до утримання та портабельність.

ISO/IEC 25000 Серія (SQuaRE): Це серія стандартів, що охоплює всі аспекти оцінювання якості ПЗ, включаючи вимоги, метрики та оцінювання.

CMMI (Capability Maturity Model Integration): Це модель розвитку, яка охоплює процеси розробки ПЗ і орієнтована на покращення процесів в організації.

IEEE 730: Цей стандарт визначає процеси управління якістю ПЗ і вимоги до гарантії якості ПЗ.

IEEE 829: Стандарт для документації тестування ПЗ, що визначає формати та зміст тестових планів, сценаріїв, випробувань та звітів.

ISTQB (International Software Testing Qualifications Board): Хоча це не є стандартом у традиційному розумінні, ISTQB визначає набір методологій та практик для тестування ПЗ, а також надає сертифікацію тестувальникам ПЗ.

Agile і Scrum: Ці методології розробки не є стандартами якості в традиційному розумінні, але вони включають практики та принципи, спрямовані на створення якісного ПЗ, шляхом постійної інтеграції, тестування та зворотного зв'язку.

Ці документи та стандарти служать основою для створення, тестування та управління якісним програмним забезпеченням. Вони встановлюють критерії якості та допомагають організаціям визначити, як вони повинні підходити до розробки ПЗ для забезпечення його відповідності вимогам якості.

1.6. ISO/IEC 25010

З перерахованих вище документів, одним з найпопулярніших та найбільш вживаних є ISO/IEC 25010, який є частиною серії стандартів ISO/IEC 25000 (SQuaRE)**ISO/IEC 25010**: Модель якості систем та програмного забезпечення. Цей стандарт визначає модель якості, що може бути застосована до будь-якого виду комп'ютерних систем, а також до окремого програмного забезпечення. Він замінив попередній стандарт ISO/IEC 9126 і розширив його, враховуючи нові аспекти якості.

ISO/IEC 25010 включає дві головні моделі:

Модель якості продукту: Ця модель охоплює характеристики якості, які є прямо пов'язані з самим продуктом. Ці характеристики поділяються на:

- Функціональна придатність
- Виконання
- Сумісність
- Надійність
- Безпека використання
- Здатність до утримання
- Портабельність

Модель якості в інтеракції: Ця модель охоплює характеристики якості, які відносяться до способу, яким користувач взаємодіє з системою. Характеристики цієї моделі включають:

- Ефективність використання
- Відповідність очікувань користувача
- Вільність від непотрібних функцій
- Задоволення користувача
- Безпека в інтеракції
- Естетичність інтерфейсу
- Доступність для всіх користувачів

Крім вищевказаних характеристик якості, стандарт також визначає метрики, які можна використовувати для виміру кожної характеристики.

ISO/IEC 25010 є ключовим для організацій, що розробляють або покупають програмне забезпечення, оскільки він допомагає встановити ясні вимоги до якості і забезпечити їх виконання.

1.7. Існуючі методи підбору ПЗ, їх переваги та недоліки

В IT-сфері існують численні методи, способи та засоби вибору програмного забезпечення. Вони різноманітні і залежать від специфіки потреби.

Експертна оцінка:

-**Переваги**: Залучення досвідчених фахівців може надати глибокий інсайт щодо конкретних потреб та якості програмного забезпечення.

-**Недоліки**: Може бути дорогою; залежність від об'єктивності і досвіду експерта.

Використання відгуків користувачів:

-**Переваги**: Відгуки від реальних користувачів можуть допомогти зрозуміти плюси та мінуси продукту в реальному використанні.

-**Недоліки**: Відгуки можуть бути суб'єктивними або маніпульованими; може бути складно визначити релевантність відгука для конкретного випадку.

Тестові версії/Демо-версії:

-**Переваги**: Дозволяє протестувати ПЗ перед купівлею та переконатися, що воно відповідає потребам.

-**Недоліки**: Тестові версії можуть бути обмеженими у функціоналі або часі використання.

Аналіз вимог:

-**Переваги**: Систематичний підхід до визначення і порівняння вимог до програмного забезпечення.

- **Недоліки**: Може бути трудомістким і потребує глибокого розуміння бізнес-процесів

Порівняльний аналіз:

-**Переваги**: Допомагає об'єктивно оцінити ПЗ, порівнюючи його з аналогами.

-**Недоліки**: Не завжди можливо знайти аналогічне ПЗ для порівняння; потребує часу та ресурсів.

Оцінка TCO (Total Cost of Ownership):

-**Переваги**: Допомагає розглянути не тільки вартість ліцензії, але й інші пов'язані витрати, такі як підтримка, навчання тощо.

-**Недоліки**: Може бути важко визначити всі складові витрат.

Оцінка ROI (Return on Investment):

-**Переваги**: Допомагає визначити економічну ефективність інвестицій у ПЗ.

-**Недоліки**: Потребує детального фінансового аналізу та прогнозування.

Ці методи можна комбінувати для отримання найбільш об'єктивної і повної картини при виборі програмного забезпечення.

1.8. Висновки до розділу 1

У сучасному IT-світі важливість правильного підбору програмного забезпечення набуває критичного значення.

Результати досліджень, таких як Forrester Wave Reports, важливі для IT-співтовариства, оскільки вони надають об'єктивний погляд на поточний стан ринку програмного забезпечення та вказують на тенденції розвитку.

Однак, незалежно від досліджень, критерії якості для ПЗ визначаються стандартами, такими як ISO/IEC 25010, який є частиною серії стандартів ISO/IEC 25000 (SQuaRE). Ці стандарти визначають атрибути якості ПЗ, як-от

функціональність, надійність, зручність використання, продуктивність, безпека та сумісність.

Методи відбору ПЗ в ІТ-сфері є різноманітними, включаючи аналіз відгуків, використання демо-версій, та оцінку технічних вимог.

У підсумку, правильний підбір ПЗ є ключовим для досягнення успіху в ІТ-проектах. Врахування потреб користувача, забезпечення якості і використання надійних джерел для вибору ПЗ може значно підвищити продуктивність та задоволеність користувачів.

РОЗДІЛ 2
ІНСТРУМЕНТИ РОЗРОБКИ ПРОЄКТУ

2.1. Важливість вибору інструметів

Правильний інструмент допоможе швидше і точніше обирати програмне забезпечення, яке відповідає конкретним вимогам ІТ-фахівців. Це може значною мірою поліпшити продуктивність роботи. Вибір невірного програмного забезпечення може призвести до непотрібних витрат на ліцензії, навчання персоналу або додаткову інтеграцію. Правильний підбір інструментів може допомогти уникнути цих витрат.

Використання відповідних інструментів може поліпшити якість кінцевого продукту, так як програмне забезпечення буде відповідати всім вимогам та стандартам. Правильно підібране програмне забезпечення легше інтегрувати з іншими системами і технологіями, які вже використовуються в організації. Використання відповідних інструментів зменшує ризик зіткнення з проблемами сумісності, витратами на технічну підтримку або проблемами безпеки. ІТ-фахівці мають конкретні потреби та вимоги до програмного забезпечення. Правильний підбір інструментів гарантує, що їх потреби будуть задоволені. Вибір гнучких і модульних інструментів може полегшити адаптацію до майбутніх технологічних змін або змін в бізнес-процесах організації.

2.2. Інтегровані середовище розробки «IDE»

IDE (Integrated Development Environment, Інтегроване середовище розробки) — це програмний комплекс, який об'єднує в собі ряд інструментів для розробки програмного забезпечення. Ці інструменти можуть включати:

1. Текстовий редактор: дозволяє розробникам писати та редагувати код.

2. Компілятор: перетворює вихідний код, написаний розробником, в машинний код або байт-код.

3. Лінкер: поєднує різні частини коду та бібліотеки в один виконуваний файл.

4. Відлагоджувач (debugger): допомагає розробникам виявляти та усувати помилки в коді, дозволяючи керувати виконанням програми, переглядати змінні та стек викликів.

5. Система управління версіями: дозволяє відслідковувати зміни в коді, зливати зміни від різних розробників і відкатувати небажані зміни.

6. Засоби автоматизації збірки: такі як Make, Maven або Gradle, дозволяють автоматизувати процес збірки коду в програму.

7. Інструменти для проектування: дозволяють розробникам графічно створювати інтерфейси, діаграми баз даних або архітектуру програми.

IDE може спростити і прискорити процес розробки, надаючи розробникам зручний інтерфейс для виконання різноманітних завдань, пов'язаних з написанням, тестуванням, збіркою і відлагодженням програмного забезпечення. Приклади популярних IDE включають Visual Code, Visual Studio, IntelliJ IDEA, та багато інших.

2.3. «IDE» проекту

Visual Studio Code

Visual Studio Code (часто абревіатура "VS Code") — це безкоштовне інтегроване середовище розробки (IDE), створене компанією Microsoft. Хоча

це є частиною сім'ї продуктів Visual Studio, VS Code є більш легким і фокусується на кодуванні, тоді як традиційне середовище Visual Studio пропонує повний набір інструментів для розробки програмного забезпечення. Основні характеристики та особливості Visual Studio Code:

- Кросплатформеність: VS Code доступний для Windows, macOS та Linux.

- Розширення: Через магазин розширень VS Code можна додавати додаткові функції для підтримки різних мов програмування, інструментів і фреймворків.

- Вбудований Git: Інтеграція з системою управління версіями Git дозволяє комітити, отримувати зміни та переглядати історію прямо з VS Code.

- Інтелектуальне автодоповнення: За допомогою функції IntelliSense VS Code пропонує автодоповнення, визначення та іншу інформацію на основі контексту коду.

- Відлагоджувач: VS Code має вбудований дебагер, що дозволяє встановлювати точки зупинки, перевіряти стан змінних та керувати виконанням коду.

- Теми: Можна налаштовувати зовнішній вигляд IDE за допомогою тем, які змінюють кольори, іконки та інші візуальні елементи.

- Термінал: Інтегрований термінал дозволяє виконувати команди шелу без виходу з середовища.

VS Code отримав велику популярність серед розробників завдяки своїй гнучкості, продуктивності та розширюваності. Його можна використовувати для розробки великого спектра програмних продуктів, від веб-сайтів до мобільних додатків і великих систем.

2.4. MySQL Workbench

MySQL Workbench — це офіційний інструмент від команди розробників MySQL для дизайну, розробки та адміністрування баз даних. Середовище надає користувачам графічний інтерфейс для роботи з базами даних MySQL, спрощуючи процеси, які можуть бути складними в командному рядку. З його допомогою можна легко створювати нові бази даних, проектувати таблиці та взаємозв'язки між ними. Крім того, MySQL Workbench містить інструменти для профілювання сервера, а також візуальні інструменти для створення SQL-скриптів. Це ідеальний вибір для тих, хто шукає графічний інструмент для роботи з MySQL.

Обрання MySQL Workbench для проекту було обґрунтовано численними перевагами, які пропонує це середовище.

MySQL Workbench є комплексним інструментом, який поєднує моделювання даних, SQL розробку і адміністрування сервера в одному пакеті. Це дозволяє вам розробляти, тестувати та розгортати бази даних з одного додатка.

З можливостями візуального проектування ви можете створювати, переглядати та редагувати моделі бази даних, використовуючи ER-діаграми, що полегшує процес розробки і розуміння структури даних.

Інструменти відлагодження дозволяють вам виявляти та вирішувати проблеми у SQL-запитах, оптимізувати їх для кращої продуктивності.

Дане середовище має вбудовані інструменти для управління MySQL сервером, включаючи моніторинг ресурсів, управління користувацькими обліковими записами та конфігурацію безпеки.

Інтегровані засоби для створення резервних копій даних і їх відновлення забезпечують захист ваших даних від втрат.

Ця IDE доступний для Windows, macOS та Linux, що дозволяє вам працювати з вашою базою даних на будь-якій платформі.

MySQL Workbench є вільним використанням, що робить його доступним для студентів, незалежних розробників та малих команд.

Ця платформа має сильну спільноту користувачів, що гарантує постійне оновлення продукту, велику кількість додаткових ресурсів, плагінів та підтримки на форумах.

У підсумку, обравши MySQL Workbench для проєкту, було отримано потужний, гнучкий та інтегрований інструмент для розробки та адміністрування бази даних.

2.5. DevTools

DevTools Chrome – це набір інструментів, вбудований безпосередньо в браузер Google Chrome, призначений для розробників веб-сайтів. Він дозволяє розробникам інспектувати, тестувати та модифікувати веб-сайти в реальному часі. З його допомогою можна аналізувати взаємодії на сторінці, вивчати структуру DOM дерева та модифікувати CSS на льоту. Засоби профілювання допомагають визначити, які частини сайту потребують оптимізації. DevTools також має спеціальні режими для симуляції роботи в мобільних браузерах, а також можливості для аналізу безпеки веб-застосунків.

DevTools (інструменти розробника), які інтегровані в більшість сучасних веб-браузерів (особливо в Chrome), є важливим інструментом для веб-розробників. Ось декілька причин, чому вибір DevTools був вдалим для вашого проекту, а також ключові переваги цього інструменту:

Інтерактивний Дебагінг DevTools дозволяє вам швидко виявляти та усувати помилки в JavaScript, допомагаючи знайти проблемне місце в коді і виправити його в реальному часі. Візуальна стилізація даного набору

інструментів дозволяють робити відживлені зміни стилізації, що спрощує процес налаштування дизайну веб-сторінки. З допомогою DevTools можна аналізувати продуктивність веб-сторінки, виявляючи елементи або скрипти, які можуть сповільнювати завантаження сторінки. Симуляція мобільних пристроїв надає можливість перегляду веб-сторінки так, як вона виглядає і працює на різних мобільних пристроях. Моніторинг мережевої активності виконується через панель "Network" можна відслідковувати всі мережеві запити, які робить веб-сторінка, що дозволяє глибше зрозуміти, як дані завантажуються та обробляються. Також, одною з переваг DevTools є інтеграція з іншими інструментами. Цей набір інструментів легко інтегрується з іншими платформами, такими як Lighthouse, для додаткового аналізу і оптимізації веб-сайтів.

Загалом, DevTools надає повний набір інструментів для ефективної та ефікасної веб-розробки, забезпечуючи глибокий аналіз, діагностику проблем та швидке внесення змін до веб-сайту в реальному часі.

2.6. Front-end

React.js

Використання React.js для розробки даного проєкту надає численні переваги. React дозволяє створювати інтерфейс на основі компонентів, полегшуючи повторне використання коду та забезпечуючи модульність. Його ключовою особливістю є віртуальний DOM, який оптимізує перемальовування інтерфейсу, роблячи додаток більш відгуковим. Додатково, завдяки великому та активному співтовариству розробників, React може похвалитися швидкою підтримкою та численними готовими рішеннями. Його гнучкість дозволяє інтегрувати React з іншими бібліотеками та фреймворками для створення складних додатків. Оскільки React підтримується Facebook, це гарантує його стабільність і надійність. Крім того,

з можливістю використовувати React Native, розробники можуть створювати нативні мобільні додатки для різних платформ, використовуючи більшість коду з веб-додатка.

TypeScript

Використання TypeScript разом із React.js для проекту відкриває ряд стратегічних переваг. По-перше, TypeScript пропонує строгу типізацію, що дозволяє зловити багато помилок на етапі компіляції, замість того, щоб вони проявилися в рантаймі. Це зменшує кількість помилок у продукті, спрощуючи його підтримку та розвиток.

З TypeScript код стає більш зрозумілим та автодокументованим завдяки типам. Це полегшує розробку, коли в команді декілька розробників, адже кожен може швидко розібратися в коді колег.

Також TypeScript інтегрується з більшістю сучасних інструментів розробки, що полегшує процес налаштування та розгортання додатку. Зокрема, він добре працює з React, надаючи можливість використовувати ще один рівень безпеки та ефективності завдяки типізації пропсів, стану компонентів і так далі.

Взагалі кажучи, комбінація React і TypeScript надає проєкту більше стабільності, зрозумілості та масштабованості, дозволяючи одночасно втілювати складні рішення з високою впевненістю в якості коду.

2.7. Back-end

Nest.js

Nest.js — це прогресивний фреймворк для побудови серверних застосунків на Node.js. Ось докладніший огляд цього фреймворку:

Хоча Nest.js може використовувати інші бібліотеки, він за замовчуванням використовує Express.js, що є однією з найпопулярніших бібліотек для побудови веб-додатків на Node.js. Це надає Nest.js змогу

використовувати всю екосистему Express.js та його плагіни. Його модульна структура забезпечує чітке розділення логіки і спрощує масштабування. Кожен модуль може мати власні сервіси, контролери та провайдери, які легко імпортуються та повторно використовуються. Одна з основних особливостей даного фреймворку— це декоратори, які дозволяють додавати метадані до класів, методів, властивостей і параметрів. Це полегшує конфігурацію і розширення функціональності додатку. Nest.js активно використовує TypeScript, надаючи розробникам всі переваги строгого типізування і сучасних особливостей ECMAScript. Як вже згадувалося, цей фреймворк підтримує мікросервісну архітектуру. Він надає гнучкі та потужні інструменти для побудови масштабованих мікросервісних систем з використанням різних транспортних протоколів. Система впровадження залежностей Nest.js допомагає забезпечити гнучкість, модульність та тестовість вашого додатку, дозволяючи вам створювати дійсно декларативні класи і методи.

Інтерсептори, фільтри винятків, та сторожові (Gard). Ці особливості дозволяють розробникам легко розширювати або змінювати поведінку додатків, вилучаючи повторюваний код або реалізуючи складну логіку обробки.

Nest.js — це потужний та гнучкий фреймворк, який дозволяє розробникам створювати ефективні, масштабовані та структуровані серверні застосунки на Node.js.

Використання Nest.js визначається кількома ключовими перевагами. Nest.js є потужним фреймворком, який базується на Express.js, забезпечуючи ефективність останнього з доданою структурою і модульністю. Це дозволяє швидко і ефективно створювати стійкі, масштабовані й добре організовані веб-додатки.

Основна відмінність Nest.js полягає в його декораторах і вбудованих інструментах залежностей. Це поліпшує гнучкість коду та його здатність до повторного використання, сприяючи створенню чистіших і більш модульних додатків.

Крім того, Nest.js пропонує підтримку мікросервісної архітектури, що може бути особливо корисним для великих і складних додатків, які потребують високої масштабованості та розділення задач.

Загалом, вибір Nest.js для back-end для цього проєкту гарантує сучасний підхід до розробки з використанням найкращих практик, що сприяє ефективності, стабільності та масштабованості вашого додатку.

TypeORM

TypeORM є однією з найпопулярніших ORM (Object-Relational Mapping) бібліотек для TypeScript і JavaScript. Використання TypeORM у цьому проєкті було хорошим рішенням з кількох причин:

По-перше, TypeORM спеціально розроблений для TypeScript, що означає, що він ідеально підходить для проєктів, які використовують TypeScript на сервері. Було отримано всі переваги строгого типізування під час роботи з вашою базою даних.

По-друге, TypeORM є дуже гнучким і підтримує багато систем управління базами даних. Це означає, що можна використовувати ту ж саму бібліотеку незалежно від того, яка СУБД була обрана, що полегшує міграцію або розширення у майбутньому.

Третім важливим моментом є активна спільнота та регулярні оновлення. Це важливо для забезпечення безпеки, внесення поліпшень та додавання нових функцій.

Крім того, TypeORM надає можливість використовувати декоратори для визначення та налаштування сутностей, що робить код більш декларативним і читабельним.

Враховуючи всі ці переваги, можна стверджувати, що вибір TypeORM для проєкту було розумним рішенням, що допоможе створити надійний, гнучкий та масштабований back-end.

2.8. База даних

MySQL є однією з найпопулярніших систем управління реляційними базами даних на світі.

На перший план виступає відмінна продуктивність та надійність MySQL. Протягом багатьох років вона була випробована величезною кількістю великих і малих проєктів, що підтверджує її здатність ефективно обробляти великі обсяги даних.

До того ж, MySQL відома своєю гнучкістю. Вона підтримує велику кількість типів даних, функцій та може бути налаштована так, щоб відповідати специфічним потребам розробки.

Однією з ключових особливостей MySQL є відкритий код. Це дозволяє спільноті вносити свій вклад у розвиток бази даних, що сприяє постійному поліпшенню продукту. Відкритий код також означає, що розробники мають повний контроль над своєю базою даних і можуть адаптувати її до своїх потреб.

З огляду на безпеку, MySQL пропонує ряд функцій, таких як шифрування даних, аутентифікація на рівні користувача та інструменти для моніторингу і аудиту, що допомагає забезпечити захист ваших даних.

Узагальнюючи, використання MySQL для вашого проєкту забезпечує комбінацію продуктивності, гнучкості, відкритого коду, безпеки та сильної підтримки спільноти, що робить її відмінним вибором для будь-якого проєкту.

2.9. Поштовий сервіс

SendGrid, є провідним постачальником послуг електронної пошти як сервісу (Email as a Service).

SendGrid надає надійність в доставці електронних листів. Важливість цього аспекту не можна недооцінювати, оскільки незалежно від того, наскільки хороший контент ви відсилаєте, якщо він не доходить до адресата, весь зусилля стає марним.

Інтеграція SendGrid з іншими платформами і фреймворками часто є досить простою завдяки їхнім API. Це означає, що ви можете легко додавати функції відправлення електронної пошти до своїх застосунків без необхідності глибокого занурення в деталі реалізації.

SendGrid надає відмінні аналітичні інструменти, які дозволяють вам відстежувати відправлені електронні листи. Ви можете дізнатися, які листи були відкриті, на які посилання було натиснуто, чи були листи видалені без прочитання і так далі. Це надає вам цінну інформацію про те, як ваші листи сприймаються користувачами.

Безпека та захист є важливими аспектами будь-якої служби електронної пошти, і SendGrid приділяє особливу увагу цим питанням. З їхньою допомогою ви можете бути впевненими, що ваші комунікації захищені і приватні.

Загалом, використання SendGrid у вашому проєкті дозволяє вам ефективно і надійно комунікувати з вашими користувачами або клієнтами, одночасно забезпечуючи простоту інтеграції, аналітичні можливості та безпеку.

2.10. Google OAuth

Google OAuth – це система автентифікації, яка дозволяє користувачам надавати вашому застосунку доступ до їхніх даних у Google без необхідності

передачі вам пароля. Це не тільки полегшує процес входу, але й забезпечує додатковий рівень безпеки, оскільки користувачам не потрібно створювати новий пароль, а ви, як розробник, не зберігаєте паролі користувачів.

Однією з головних переваг Google OAuth є його широка підтримка та впізнаваність. Багато користувачів вже мають акаунти Google, тому дозволяючи їм використовувати ці акаунти для входу, ви зменшуєте бар'єри для реєстрації та входу у вашому застосунку. Коли користувач бачить опцію входу через Google, він може відчувати додаткову впевненість у безпеці процесу, оскільки довіряє Google як надійному провайдеру.

Додатково, Google OAuth включає різноманітні засоби безпеки, такі як двофакторна аутентифікація. Якщо користувач включив цю функцію в своєму акаунті Google, ваш застосунок автоматично отримує ці переваги без додаткових зусиль з вашого боку.

Також не можна забувати про простоту інтеграції. Google надає чіткі документації та бібліотеки для різних мов програмування, що дозволяє легко додати OAuth-автентифікацію до вашого застосунку.

Загалом, вибір Google OAuth для проєкту не тільки підвищує зручність для користувачів, але й забезпечує вищий рівень безпеки та довіри до застосунку.

2.11. Хмарні сервіси AWS

AWS, або Amazon Web Services, є комплексом хмарних сервісів від компанії Amazon, який надає засоби для розгортання, управління та масштабування застосунків у Інтернеті. Він пропонує широкий спектр інструментів від обчислення і зберігання даних до аналітики та машинного навчання. AWS надає можливість користувачам "орендувати" комп'ютерні ресурси без потреби купувати та управляти фізичним обладнанням. Завдяки

гнучкості, швидкості та безпеці, які надає AWS, багато компаній вибирають цю платформу для своєї хмарних інфраструктури. AWS завоював репутацію одного з лідерів у сфері хмарних технологій завдяки своїм інноваційним рішенням та широкому спектру послуг. Конкретно в даному проекті була використана низка сервісів Amazon, перелік з описом буде наведено нижче.

AWS Elastic Beanstalk є відмінним сервісом для розгортання, управління та масштабування веб-застосунків та служб. Ви просто завантажуєте свій код, а Elastic Beanstalk автоматично обслуговує процес розгортання, від моніторингу до масштабування та завантаження балансувальника. Таким чином, ви можете зосередитися на коді, не переймаючись інфраструктурою.

AWS RDS стає величезною допомогою, коли мова йде про роботу з реляційними базами даних. Сервіс автоматизує завдання, такі як резервне копіювання, оновлення ОС та баз даних, а також налаштування реплікації для підвищення доступності та стійкості до відмов. Використовуючи RDS, ви можете забезпечити надійне, швидке та масштабоване рішення для управління базами даних.

AWS Route 53 забезпечує вас надійним та масштабованим способом перенаправляти користувачів на інтернет-інфраструктуру вашого застосунку. Це не лише служба доменних імен, але й розподілена мережа доставки контенту, яка забезпечує низьку затримку і високу доступність.

Об'єднуючи ці послуги, була отримана , масштабована та ефективна інфраструктура для застосунку. AWS дозволяє використовувати передові технології в хмарних обчисленнях, звільняючи розробника від завдань, пов'язаних з управлінням фізичною інфраструктурою, та дозволяє зосередитися на створенні програмного забезпече

2.12. User flow

User flow (потік користувача) – це візуалізація послідовності кроків, які користувач робить під час взаємодії з продуктом, сайтом чи застосунком. Цей термін часто використовується в дизайні користувацького досвіду (UX) для планування та створення логічної, зрозумілої та ефективної навігації для користувачів.

User flow допомагає командам зрозуміти, як користувачі будуть переміщатися по продукту, де вони можуть стикнутися з перешкодами та які дії вони виконуватимуть під час цього процесу. Це може включати все від того, як користувач знаходить сайт або додаток, до взаємодії з конкретними функціями та завершення певних завдань.

Для створення ефективного user flow дизайнери часто використовують різні інструменти, такі як схеми, діаграми та прототипи, які дозволяють візуально представити шлях користувача через продукт і ідентифікувати потенційні проблеми або підвищити ефективність взаємодії.

User flow Реєстрація Користувача

1. Користувач: Схема починається з користувача, який є вихідною точкою для всього потоку.

2. Перехід на веб-сторінку: Користувач переходить за посиланням на веб-сайт.

3. Форма реєстрації: На цьому етапі користувачу пропонується форма, де він повинен ввести свою електронну пошту, щоб отримати одноразовий код перевірки.

4. Відправка коду перевірки: Після введення адреси електронної пошти користувачеві відправляється код для перевірки.

5. Введення коду перевірки: Користувач повинен ввести отриманий код для подальшої верифікації.

6. Перевірка коду: Система перевіряє правильність введеного користувачем коду.

7. Заповнення особистої інформації: Якщо код вірний, користувачу пропонується форма для введення додаткової інформації: назви компанії, посада, використовуваних технологій, стаж роботи.

8. Основна сторінка: Після завершення реєстрації та заповнення всієї необхідної інформації користувач переходить на основну сторінку сайту.

Дана блок-схема іллюструє процес реєстрації користувача на веб-сайті з використанням коду перевірки, що відправляється на електронну пошту.

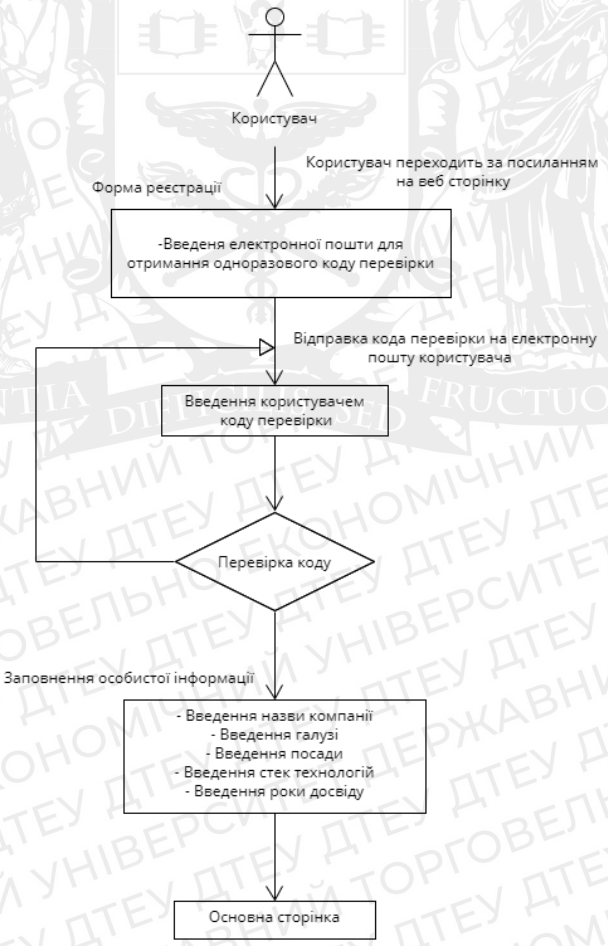

Рис. 2.1. User flow користувача при єрестрації

Джерело: розроблено автором

2.13. User Flow Функціоналу програми

Користувач розпочинає свій шлях на веб-ресурсі, починаючи з Основної сторінки. З цієї сторінки у нього є кілька можливостей дій:

1. Він може перейти на **Сторінку профілю користувача**.

Внести **Зміни до особистих даних**. Після внесення змін користувач може повернутися назад на Основну сторінку або вийти з ресурсу, використовуючи опцію **Вихід з облікового запису**.

2. З Основної сторінки користувач також може перейти на **Сторінку історії запропонованого ПЗ**. Відтак, він може повернутися назад на Основну сторінку.

3. Крім цього, користувач може перейти на **Перегляд запропонованого ПЗ**. Тут він має можливість:

Здійснити **Зміну параметрів підбору ПЗ**. Після цього, повернутися до Перегляду запропонованого ПЗ або ж вийти з ресурсу через опцію Вихід з облікового запису.

На кожному етапі, незалежно від того, де знаходиться користувач, у нього є можливість вийти з ресурсу, використовуючи опцію **Вихід з облікового запис**у.

Ця блок-схема відображає користувацький досвід на веб-ресурсі, починаючи з головної сторінки і закінчуючи різними варіантами взаємодії користувача з ресурсом, включаючи перегляд профілю, історії та налаштувань програмного забезпечення.

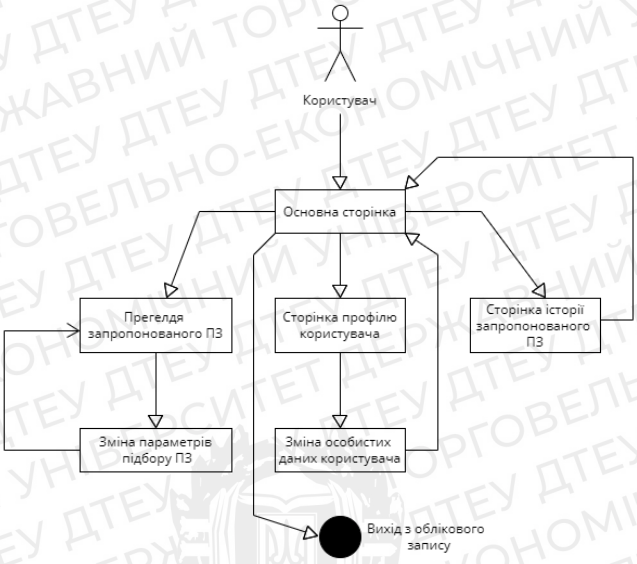

Рис 2.2. User flow користувача на головній сторінці

Джерело: розроблено автором

2.4. Висновки до розділу 2

У сучасному світі правильний підбір інструментів для створення ПЗ є критично важливим, оскільки він безпосередньо впливає на продуктивність розробки, якість кінцевого продукту та його масштабованість. Вибір невідповідних інструментів може призвести до збільшення витрат часу, ресурсів та коштів. Водночас правильно обрані технології дозволяють командам швидко адаптуватися до змін, враховуючи потреби ринку. Крім того, вони забезпечують безпеку, надійність та відмінний користувацький досвід, що є ключовим для успіху будь-якого сучасного ПЗ.

В даному розділі було наведено перелік інструментів та ресурсів які використовувались під час проектування та розробки Професійно-орієнтованої експертної системи підбору програмного забезпечення за запитами ІТ-фахівців. Обґрунтуванням вибору даних інструментів та програм був детальний опис технологій та переваг кожної з них

РОЗДІЛ 3
ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ ЕКСПЕРТНОЇ СИСТЕМИ

3.1. Загальна характеристика інформаційного забезпечення

Вебсайт експертної системи представляє собою комплексний веб-ресурс, призначений для допомоги IT-фахівцям у виборі програмного забезпечення. Основні елементи складу включають:

-**Базу даних програмного забезпечення:** централізоване сховище інформації про різне програмне забезпечення, їх характеристики, відгуки, рейтинги тощо.

-**Інтерфейс користувача:** графічний інтерфейс, що дозволяє користувачам взаємодіяти з системою, вводити запити та отримувати рекомендовані варіанти ПЗ.

-**Алгоритми обробки запитів:** набір логічних процедур та методів, які аналізують запит користувача і вибирають найбільш підходящі варіанти програмного забезпечення з бази даних.

-**Структура:**

Проект структурований з урахуванням модульного підходу:

- Фронтенд: Відповідає за представлення інформації користувачеві. Зазвичай це веб-інтерфейс, який користувач бачить у своєму браузері.

- Бекенд: Забезпечує обробку запитів, взаємодію з базою даних і реалізацію бізнес-логіки.

- База даних: Місце зберігання всієї інформації про програмне забезпечення, відгуки, рекомендації тощо.

- **Принципи організації:**

Система базується на кількох ключових принципах:

- Модульність: Кожен компонент системи розроблений так, щоб його можна було модифікувати або замінювати без втрати функціональності інших частин.

- Масштабованість: Система може легко адаптуватися до збільшення кількості користувачів або обсягу даних.

3.2. Тип носія даних:

База даних використовує MySQL, яка розміщена на сервісі Amazon RDS (Relational Database Service). Amazon RDS дозволяє легко налаштовувати, використовувати та масштабувати реляційну базу даних у хмарі. Він також забезпечує резервне копіювання даних, автоматичні патчі та автоматичне відновлення даних.

Обґрунтування вибору технологій

- **Amazon RDS**

- Ефективність: Amazon RDS автоматично робить резервні копії бази даних, виконує патчі ОС та бази даних і відновлює базу даних після збоїв.

- Масштабованість: Легкість масштабування ресурсів відповідно до потреб, з допомогою їх автоматичного або ручного масштабування.

- Безпека: Вбудовані механізми безпеки, включаючи автоматичне шифрування даних.

- **AWS Elastic Beanstalk**

- Подвійна користь: Elastic Beanstalk автоматично керує розгортанням, від масштабування додатку до балансування навантаження та моніторингу.

- Легкість використання: Вам потрібно лише завантажити свій код, а Elastic Beanstalk автоматично виконає весь процес розгортання, від запуску серверів до налаштування баз даних.

- Інтеграція з Amazon RDS: Забезпечує безшовну інтеграцію з Amazon RDS, що спрощує налаштування, моніторинг та масштабування баз даних.

- Опис прийнятих методів контролю інформації

Різноманітні інструменти AWS використовуються для моніторингу та контролю ресурсів. Зокрема, Amazon CloudWatch для моніторингу продуктивності додатків та баз даних, а також AWS Identity and Access Management (IAM) для контролю доступу до ресурсів.

- Вимоги до надійності і достовірності інформації:

Завдяки використанню AWS, наша система забезпечує високу надійність і доступність. RDS гарантує стабільність роботи бази даних, а Elastic Beanstalk гарантує неперервну роботу нашого додатку.

3.3. Методи контролю інформації:

В контексті професійно-орієнтованої експертної системи підбору програмного забезпечення, контроль якості та достовірності інформації має вирішальне значення.

Автоматична перевірка валідності даних:

Цей процес передбачає автоматичну перевірку введення даних на клієнті та на сервері для забезпечення їх правильності, повноти та відповідності встановленим стандартам.

Валідація на клієнті (Frontend)

Використання TypeScript (React) для валідації даних перед їх надсиланням на сервер. Це може включати в себе перевірку формату електронної пошти, сили пароля, обов'язкові поля тощо.

Переваги: Швидке відгук, менше неправильних запитів до сервера.

Валідація на сервері (Backend)

Використання мови програмування сервера (Nest.js) для перевірки валідності даних, отриманих від клієнта. За допомогою вбудованих функцій MySQL на Amazon RDS створюється обмеження та тригери, які автоматично перевіряють дані на валідність перед їх додаванням або оновленням в базі даних.

Переваги: Додатковий рівень безпеки; навіть якщо зловмисники обходять валідацію на клієнті, серверна валідація забезпечить перевірку даних та запобігання додаванню невірних, неконсистентних або дубльованих даних, що підвищує надійність і якість інформації.

Аудит змін інформації

Система веде журнал всіх змін у базі даних, включаючи додавання, оновлення або видалення записів. Використовуючи AWS RDS, активується журналування для нашої бази даних MySQL. Це дозволяє відслідковувати всі зміни, які відбулися, та ідентифікувати здійснені дії та користувачів, які їх виконали.

Переваги: Забезпечення прозорості і відслідковуваності всіх дій в системі, що допомагає виявити аномалії та неправильні дії.

Регулярні бекапи бази даних

Щоб гарантувати безпеку інформації та можливість відновлення в разі аварій, система регулярно створює резервні копії бази даних. За допомогою служб AWS RDS ми налаштовуємо автоматичне резервне копіювання бази даних MySQL. Amazon RDS автоматично створює резервні копії даних, які можна використовувати для відновлення бази даних.

Переваги: Забезпечення надійності та захисту від втрати даних, є можливість швидко відновити інформацію у випадку збоїв або втрати даних.

Ці методи забезпечують, що інформація в системі завжди актуальна, достовірна та захищена від потенційних ризиків або загроз.

3.4.Вимоги до надійності і достовірності:

Забезпечення надійності і достовірності є вітальною частиною нашої професійно-орієнтованої експертної системи підбору програмного забезпечення.

Висока доступність серверів

Гарантія того, що веб-сервіс завжди буде доступний для користувачів, незважаючи на можливі збої або технічні проблеми.

Імплементація:

-Використання Amazon Elastic Beanstalk для автоматичного масштабування ресурсів відповідно до навантаження.

-Використання множинності зон доступності на AWS, щоб гарантувати, що якщо одна зона відмовить, інша зможе продовжити роботу.

-Регулярний моніторинг і алертинг за допомогою AWS CloudWatch.

Захист від DDoS атак:

Захист нашого веб-сервісу від зловмисних атак, які мають на меті завалити систему великою кількістю запитів.

Імплементація:

-Використання AWS Shield - служби, яка надає захист від DDoS атак.

-Інтеграція з системами виявлення і запобігання вторгненням.

-Ліміт на частоту запитів до API з використанням Nest.js, щоб запобігти швидким послідовним запитам.

Шифрування даних:

Забезпечення конфіденційності і цілісності даних користувачів шляхом шифрування.

Імплементація:

-Використання протоколу SSL/TLS для шифрування даних, які передаються між клієнтом (React.js з Typescript) та сервером (Nest.js).

-Шифрування даних на диску в AWS RDS за допомогою AWS Key Management Service (KMS).

-Використання секретних ключів і алгоритмів шифрування на рівні застосунку для додаткового захисту чутливих даних, таких як паролі користувачів.

Завдяки цим методам і використанню відповідних технологій забезпечується висока надійність, доступність та безпеку даних у системі.

3.5. Джерела і носії інформації:

Для забезпечення ефективної роботи нашої експертної системи підбору програмного забезпечення необхідно мати чітке уявлення про джерела і носії інформації, що використовуються.

Джерела інформації

Сайти додатків:

Веб-сайти різних програмних продуктів та додатків, які надають актуальну інформацію про характеристики, відгуки, ціноутворення та інші важливі аспекти програмного забезпечення.

Використання: Веб-скрапінг для отримання необхідної інформації з цих джерел.

Особисті дані користувачів:

Дані, надані користувачами під час реєстрації, взаємодії або використання нашої системи, включаючи переваги, вимоги до програмного забезпечення та іншу важливу інформацію.

Використання: Дані зберігаються в базі даних MySQL на AWS RDS і використовуються для персоналізації рекомендацій та підвищення якості обслуговування користувачів.

Носії інформації

Хмарні сервери:

Віртуальні сервери, розміщені в облаках, зокрема на платформі AWS, що забезпечують зберігання, обробку та доступ до даних.

Використання: Всі основні дані та бізнес-логіка застосунку зберігаються і обробляються на цих серверах. AWS Elastic Beanstalk забезпечує автоматичне масштабування та управління ресурсами.

Локальні кеші користувачів:

Тимчасові файли та дані, збережені на пристроях користувачів, щоб підвищити швидкість завантаження та зменшити завантаження на сервер.

Використання: З допомогою технології React.js з Typescript можна використовувати локальне сховище (PersistStorage) для зберігання часто використовуваної інформації, зокрема налаштувань користувача, деяких даних для офлайн-режиму або тимчасових даних, що чекають на завантаження.

Розуміння цих джерел і носіїв дозволяє оптимізувати збір, зберігання та обробку інформації, що, в свою чергу, поліпшує якість та швидкість надання послуг користувачам.

На даній діаграмі представлена структура бази даних, що включає в себе наступні основні сутності:

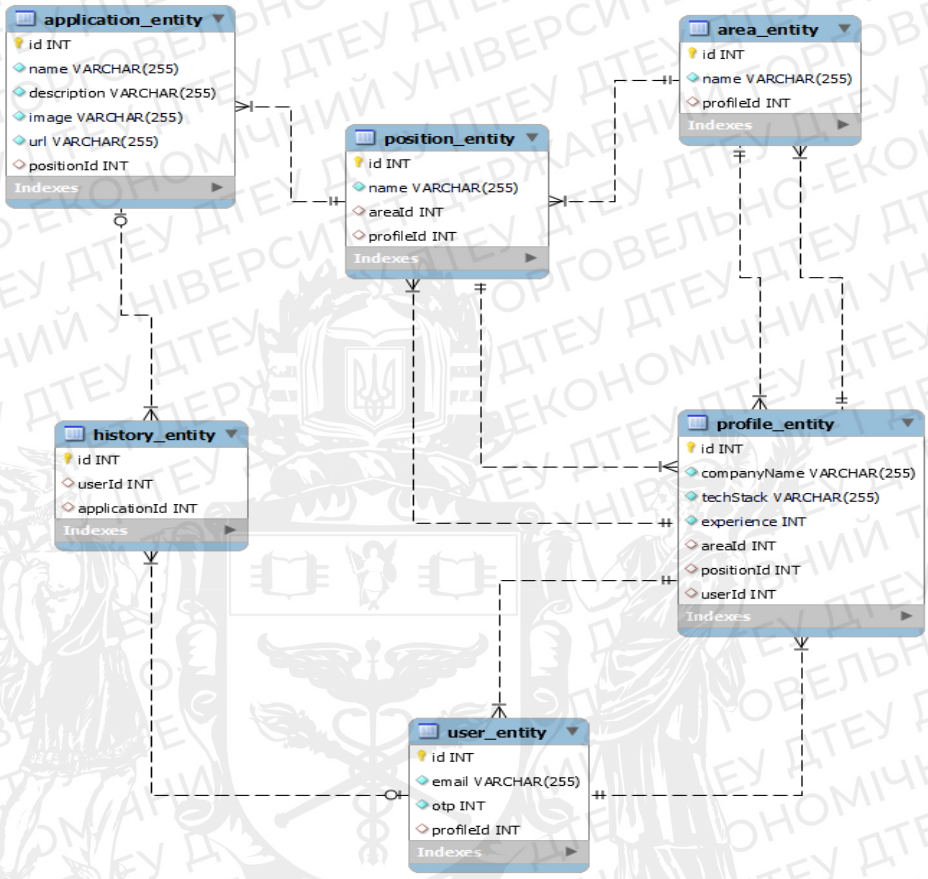

Рис. 3.1. Структура бази даних

Джерело: розроблено автором

1. application_entity:

- id: Унікальний ідентифікатор додатка.
- name: Назва додатка.
- description: Опис додатка.
- image: URL-адреса зображення додатка.
- url: URL-адреса додатка.
- positionId: Ідентифікатор пов'язаної позиції.

2. position_entity:

- id: Унікальний ідентифікатор позиції.
- name: Назва позиції.
- areaId: Ідентифікатор пов'язаної області.
- profileId: Ідентифікатор пов'язаного профілю.

3. area_entity:

- id: Унікальний ідентифікатор області.
- name: Назва області.

4. profile_entity:

- id: Унікальний ідентифікатор профілю.
- companyName: Назва компанії.
- techStack: Технологічний стек.
- experience: Досвід (у роках).
- areaId: Ідентифікатор пов'язаної області.
- positionId: Ідентифікатор пов'язаної позиції.
- userId: Ідентифікатор пов'язаного користувача.

5. history_entity:

- id: Унікальний ідентифікатор запису історії.
- userId: Ідентифікатор користувача, який взаємодієв з додатком.
- applicationId: Ідентифікатор додатка, з яким взаємодіяв користувач.

6. user_entity:

- id: Унікальний ідентифікатор користувача.
- email: Електронна адреса користувача.
- otp: Одноразовий пароль або токен для авторизації.
- profileId: Ідентифікатор пов'язаного профілю користувача.

Зв'язки між сутностями визначаються на основі ідентифікаторів та відображаються стрілками на діаграмі. Зокрема, можна зазначити зв'язки між користувачами та їх профілями, профілями та областями/позиціями, а також

між додатками та позиціями. Ця структура дозволяє гнучко взаємодіяти з різними видами інформації та надає можливість розширення в майбутньому.

3.6. Висновок до розділу 3

В даному розділі були описані ключові аспекти планування, розробки та впровадження системи, зокрема:

Методи контролю інформації

- Автоматична перевірка валідності даних.
- Аудит змін інформації.
- Регулярні бекапи бази даних.

Вимоги до надійності і достовірності

- Висока доступність серверів.
- Захист від DDoS атак.
- Шифрування даних.

Джерела та носії інформації

- Джерела: сайти додатків, особисті дані користувачів.
- Носії: Облачні сервери, локальні кеші користувачів.

Структура бази даних

Було представлено структуру бази даних та її основні сутності, такі як користувачі, профілі, додатки, позиції, області та історія взаємодії користувачів з додатками.

Для реалізації вказаних вимог та методів було обрано технології такі як React.js з Typescript для клієнтської частини, та Nest.js для серверної частини. База даних орієнтована на взаємодію з різними типами даних, що відображено у її сутностях та зв'язках.

Загалом, дана система була розроблена з урахуванням потреб користувачів, вимог безпеки та забезпечення якісної роботи додатку._web-сайту.

РОЗДІЛ 4
ОПИС РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Загальна Характеристика Програмного Забезпечення

Ціль проекту

Створення професійно-орієнтованої експертної системи, яка забезпечує оптимальний підбір програмного забезпечення на основі вимог ІТ-фахівців. Дана система покликана спростити процес вибору відповідних ІТ-рішень, зменшити час на дослідження та аналіз ринку програмного забезпечення, а також підвищити ефективність використання ІТ-ресурсів в професійній діяльності.

Завдання проекту

Аналіз вимог: Систематизувати вимоги ІТ-фахівців до програмного забезпечення в різних напрямках їхньої діяльності.

Розробка системи: Створити систему для аналізу вимог користувачів та визначення найбільш відповідного програмного забезпечення.

Створення бази даних: Зібрати базу даних сучасного програмного забезпечення, включаючи їх характеристики, можливості, та посилання на самі додатки.

Ця система буде не лише інструментом для підбору ПЗ, але й важливим елементом, який допомагає ІТ-спільноті залишатися в курсі нових технологічних рішень та оптимізувати свою роботу.

Опис основних функцій та можливостей системи.

На основі представлених вами блок-схем, основні функції та можливості вашої системи включають.

Реєстрація користувача: Система передбачає процес реєстрації, під час якого користувач вводить свою електронну пошту, отримує код перевірки на вказану адресу та вводить його для завершення реєстрації.

Заповнення особистої інформації: Після реєстрації користувачу пропонується ввести додаткову інформацію про себе, таку як назва компанії, країна, посада, використовувані технології, роки досвіду тощо.

Доступ до основної сторінки: Після реєстрації та введення інформації користувач потрапляє на основну сторінку, де може взаємодіяти з іншими функціями системи.

Перегляд запропонованого ПЗ: Користувачу пропонуються варіанти програмного забезпечення на основі його потреб та інтересів.

Зміна параметрів підбору ПЗ: Користувач може коригувати параметри для більш точного підбору рекомендованого ПЗ.

Перегляд профілю: Користувач може переглядати та редагувати свій профіль, змінюючи особисті дані або іншу інформацію.

Перегляд історії запропонованого ПЗ:** Система надає можливість переглядати історію рекомендацій ПЗ, що були запропоновані користувачу.

Вихід з облікового запису: Користувач має можливість вийти зі свого облікового запису в системі.

Ці функції та можливості роблять систему інтерактивною та адаптивною до потреб користувача і здатною забезпечувати точний підбір програмного забезпечення для ІТ-фахівці

4.2. Структура програмного забезпечення

Структура клієнтської частини

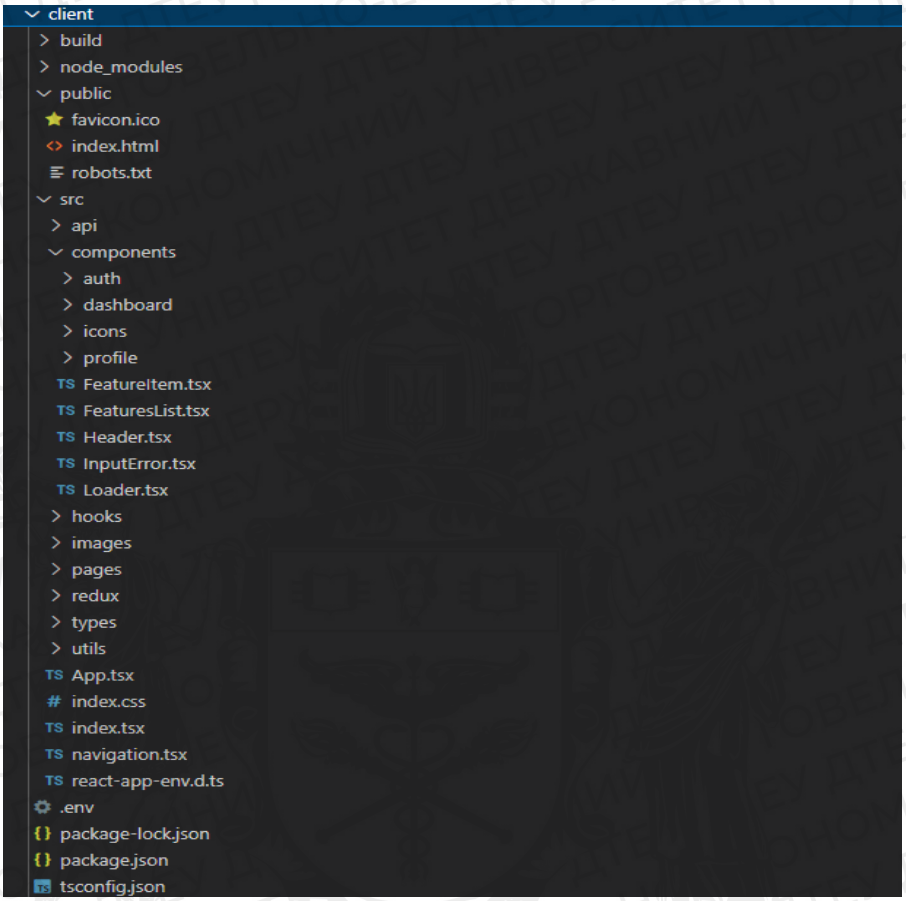

Рис. 4.1. Структура клієнстської частини коду

Джерело: розроблено автором

/build: Ця папка містить оптимізовані версії файлів, які генеруються після процесу збірки (build). Ці файли готові до розгортання на сервері.

/node_modules: Папка, яка містить усі зовнішні бібліотеки та пакети, які використовуються в проекті. Ці залежності встановлюються за допомогою менеджера пакетів npm або yarn.

/public:

favicon.ico: Іконка сайта, яка відображається у вкладці браузера.

index.html: Головний HTML файл, який слугує точкою входу для React-додатка.

robots.txt: Файл для інструкцій пошуковим роботам щодо індексації сайту.

/src: Головна папка, що містить вихідний код клієнтської частини:

/api: Папка, де зазвичай розташовані файли для взаємодії з API сервера.

/components: Компоненти React, які використовуються в проекті. Тут можна побачити різні підпапки, що відображають логічні розділи додатка, такі як auth (автентифікація), dashboard (панель керування), icons, profile тощо.

/hooks: Кастомні хуки React, які можуть використовуватися для реюзабельної логіки між компонентами.

/images: Графічні ресурси, що використовуються в додатку.

/pages: Головні "сторінки" або маршрути додатка.

/redux: Код, пов'язаний з управлінням станом додатка за допомогою Redux.

/types: TypeScript типи або інтерфейси.

/utils: Допоміжні функції та утиліти.

App.tsx: Головний компонент React додатка.

index.tsx: Точка входу для React-додатка. Звідси відбувається рендеринг App.tsx.

navigation.tsx: Можливо, код для управління навігацією або маршрутизацією в додатку.

.env: Файл для змінних середовища, який містить конфігураційні налаштування.

package-lock.json та package.json: Файли, що містять інформацію про залежності та конфігурації проекту.

tsconfig.json: Файл налаштувань для TypeScript.

Ця структура представляє собою типову організацію проекту на React і TypeScript. Вона спроектована таким чином, щоб було зручно працювати з кодом, а також забезпечувати легке масштабування проекту в майбутньому.

Структура бекенду:

/src: Головна папка, що містить вихідний код серверної частини.

/auth: Модуль автентифікації.

/dto: Data Transfer Objects - класи, які використовуються для передачі даних між методами та службами.

/guards: Захисти (guards) для рутів, які перевіряють аутентифікацію або інші умови перед наданням доступу.

/interfaces: Інтерфейси для типів даних або функцій, специфічних для автентифікації.

/strategies: Стратегії автентифікації, наприклад, JWT або OAuth.

auth.controller.ts: Контролер для обробки HTTP-запитів, пов'язаних з автентифікацією.

auth.module.ts: Модуль NestJS для автентифікації.

auth.service.ts: Служба, що містить логіку автентифікації.

/helpers: Утиліти або допоміжні функції, які можуть використовуватися в різних частинах проекту.

/mail: Модуль для роботи з електронною поштою.

/token: Модуль для роботи з токенами, наприклад, для автентифікації або відновлення пароля.

/user: Модуль користувача.

/dto: Data Transfer Objects для користувача.

/entity: Сутності, пов'язані з користувачем, наприклад, моделі бази даних.

user.controller.ts: Контролер для обробки HTTP-запитів, пов'язаних з користувачами.

user.module.ts: Модуль NestJS для користувача.

user.service.ts: Служба, що містить логіку обробки даних користувача.

app.controller.ts: Головний контролер додатка.

app.module.ts: Головний модуль NestJS додатка.

app.service.ts: Головна служба додатка.

main.ts: Точка входу для серверного додатка. Тут ініціалізується основний модуль.

.env: Файл для змінних середовища, який містить конфігураційні налаштування.

.eslintrc.js: Налаштування для ESLint, інструмента для аналізу коду на наявність проблем.

.gitignore: Файл, який містить перелік файлів або папок, які не повинні бути включені до репозиторію Git.

.prettierrc: Налаштування для Prettier, інструмента для форматування коду.

nest-cli.json: Конфігураційний файл для CLI NestJS.

package-lock.json та package.json: Файли, що містять інформацію про залежності та конфігурації проекту.

Procfile: Файл, який використовується платформами як Heroku для визначення, як запустити ваш додаток.

README.md: Файл з інструкціями та інформацією про проект.

tsconfig.build.json та tsconfig.json: Файли налаштувань для TypeScript.

Ця структура відображає типовий проект на NestJS і TypeScript. Вона організована за модулями, що дозволяє легко додавати нові функціональності та масштабувати додаток.

4.3 Основні функціональні частини ПЗ

Модуль авторизації

```
async login({ email }: LoginUserDto) {
  try {
    const _user = await this.userRepository.findOne({ where: { email } });
    const otp = generateRandomNumber();
    if (!_user) {
      await this.userRepository.save({
        email,
        otp,
      });

      await this.mailService.sendOTPEmail(email, otp);
      return 'Email has been sended';
    }

    await this.userRepository.update(_user.id, { otp });
    await this.mailService.sendOTPEmail(email, otp);
    return 'Email has been sended';
  } catch (error) {
    console.log(error);

    throw new BadRequestException(error.message, error.message);
  }
}
```

Рис. 4.2. Програмний коду модулю авторизації

Джерело: розроблено автором

Це асинхронний метод login, який приймає об'єкт з полем email типу LoginUserDto. Отже, цей метод відповідає за автентифікацію користувача за допомогою email та OTP. Якщо користувач існує, йому відправляється новий OTP. Якщо користувача немає, створюється новий запис користувача і йому також відправляється OTP.

```
async checkOTP({ email, otp }: checkUserOtpDto) {
  try {
    const _user = await this.userRepository.findOne({
      where: { email },
      relations: ['profile', 'profile.area', 'profile.position']
    });

    if (!_user)
      return new BadRequestException(
        'User is not found',
        'User is not found',
      );

    if (_user.otp !== otp)
      return new BadRequestException('OTP wrong', 'OTP wrong');

    await this.userRepository.update(_user.id, { otp: null });

    return {
      access_token: this.tokenService.generateAccessToken({
        id: _user.id,
        email: _user.email,
        profile: {
          companyName: _user.profile?.companyName,
          area: _user.profile?.area,
          position: _user.profile?.position,
          techStack: _user.profile?.techStack.split(',') as string[],
          experience: _user.profile?.experience,
        }
      }),
    };
  } catch (error) {
    throw new BadRequestException(error.message, error.message);
  }
}
```

Рис. 4.3. Програмний код методу checkOTP, який перевіряє одноразовий пароль (OTP) користувача
Джерело: розроблено автором

1. Спочатку метод шукає користувача в репозиторії за його email і отримує додаткову інформацію про його профіль.
2. Якщо користувач не знайдений, повертається помилка "User is not found".
3. Якщо введений OTP не співпадає з OTP користувача, повертається помилка "OTP wrong".
4. Якщо OTP вірний, він очищується в репозиторії.
5. Нарешті, метод повертає access_token, який генерується на основі інформації про користувача.
6. Всі помилки, які можуть виникнути у процесі, перехоплюються і повертаються як BadRequestException.

```
export class JwtAuthGuard implements CanActivate {
  async canActivate(context: ExecutionContext) {
    const request = context.switchToHttp().getRequest();
    const token = this.extractTokenFromCookies(request);

    if (!token) {
      throw new UnauthorizedException('Unauthorized');
    }
    try {
      const validate: any = jwt.verify(token, 'secret');
      request.user = validate.id;

      return true;
    } catch (e) {
      throw new UnauthorizedException(e.message);
    }
  }

  private extractTokenFromCookies(request: any): string | null {
    const cookieHeader = request.headers.cookie;
    if (cookieHeader) {
      const cookies = cookieHeader.split('; ');
      const tokenCookie = cookies.find((cookie) =>
        cookie.startsWith('access_token='),
      );
      if (tokenCookie) {
        return tokenCookie.split('=')[1];
      }
    }
    return null;
  }
}
```

Рис. 4.4. Програмний код класу JwtAuthGuard, який реалізує механізм захисту на основі JWT (JSON Web Token)
Джерело: розроблено автором

1. Метод canActivate визначає, чи має користувач доступ до певного ресурсу. Він витягує токен з cookies запиту.
2. Якщо токен відсутній, генерується виключення "Unauthorized".
3. Якщо токен присутній, він перевіряється за допомогою jwt.verify. Якщо токен вірний, id користувача зберігається в запиті, а метод повертає true.
4. У разі невірного токену генерується виключення "Unauthorized".
5. Метод extractTokenFromCookies служить для витягування токену з cookies запиту.

```typescript
export class MailService {
  constructor(private readonly configService: ConfigService) {
    SendGrid.setApiKey(this.configService.get('SENDGRID_API_KEY'));
  }

  async sendOTPEmail(email: string, OTP: number) {
    try {
      const msg = {
        to: email,
        from: {
          email: this.configService.get('SENDGRID_SENDER'),
          name: 'AppChooser',
        },
        subject: 'Auth OTP password',
        text: `Here is your password`,
        html: `<p>One-time password: ${OTP}</p>`,
      };
      return await SendGrid.send(msg);
    } catch (error) {
      throw new BadRequestException(error.message);
    }
  }
}
```

Рис. 4.5. Програмний код сервіс MailService відправляє електронні листи: Джерело: розроблено автором

1. Під час ініціалізації сервісу встановлюється API ключ для SendGrid за допомогою ConfigService.
2. Метод sendOTPEmail відправляє лист з одноразовим паролем (OTP) на вказану електронну адресу.
3. Лист має конкретного відправника, тему та основне повідомлення, що містить OTP.
4. У разі виникнення помилок генерується виключення "BadRequestException".

```typescript
async getareaposition() {
  return await this.areaRepository.find({relations: ['position']});
}

async getapps(areaId: number, positionId: number) {
  const apps = await this.areaRepository.createQueryBuilder("area")
  .innerJoinAndSelect("area.position", "position")
  .innerJoinAndSelect("position.application", "application")
  .where("position.id = :positionId AND area.id = :areaId", { positionId, areaId })
  .getMany();

  return apps[0]?.position[0]?.application;
}
```

Рис. 4.6. Програмний код методів getareaposition() та getapps(areaId, positionId) Джерело: розроблено автором

Код включає в себе два методи:

1. getareaposition(): повертає всі області (areas) з пов'язаними посадами (positions).
2. getapps(areaId, positionId): знаходить застосунки (applications) для конкретної області та посади, використовуючи об'єднання таблиць, та повертає ці застосунки.

4.4. Інтерфейс користувача

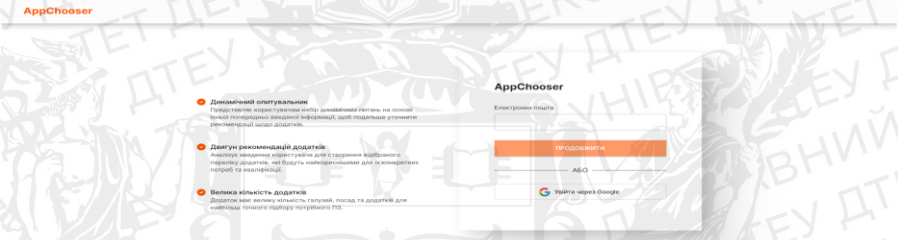

Рис. 4.7. Сторінка авторизації

Джерело: розроблено автором

На даній сторінці користувач вводить адресу своєї електронної пошти для отримання одноразового кода авторизації.

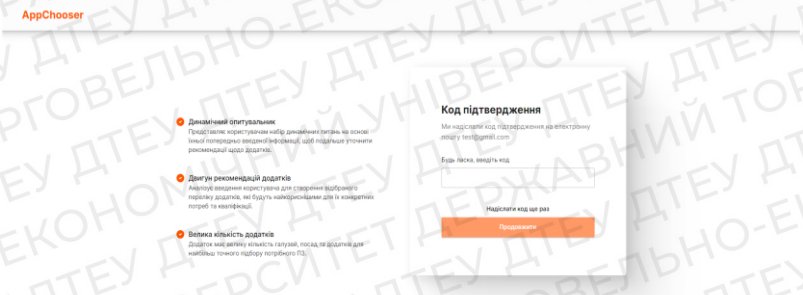

Рис. 4.8. Сторінка з кодом авторизації

Джерело: розроблено автором

На даній сторінці користувач має ввести одноразовий код авторизації який отримав на свою електронну пошту.

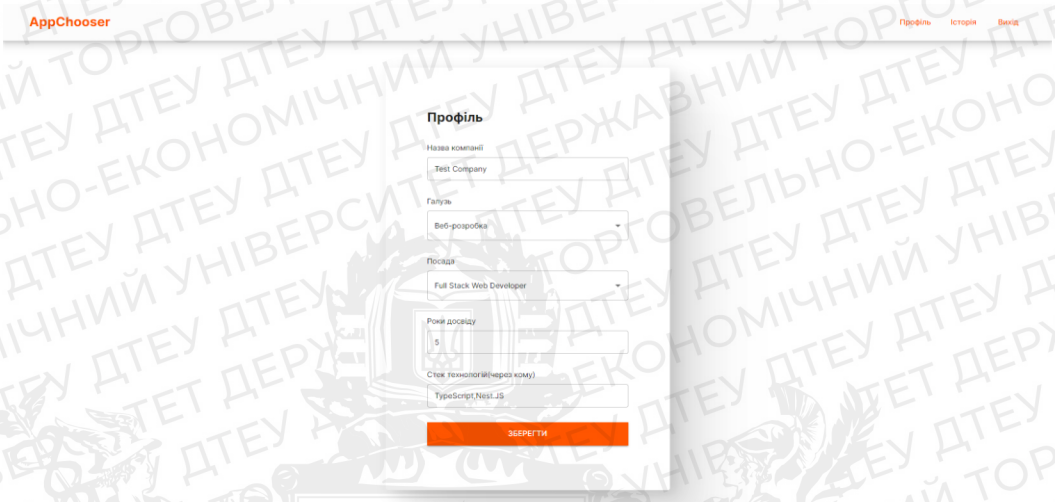

Рис. 4.9. Сторінка профілю користувача
Джерело: розроблено автором

На даній сторінці користувач вводить особисту інформацію яка далі буде використовуватись для підбору програмного забезпечення.

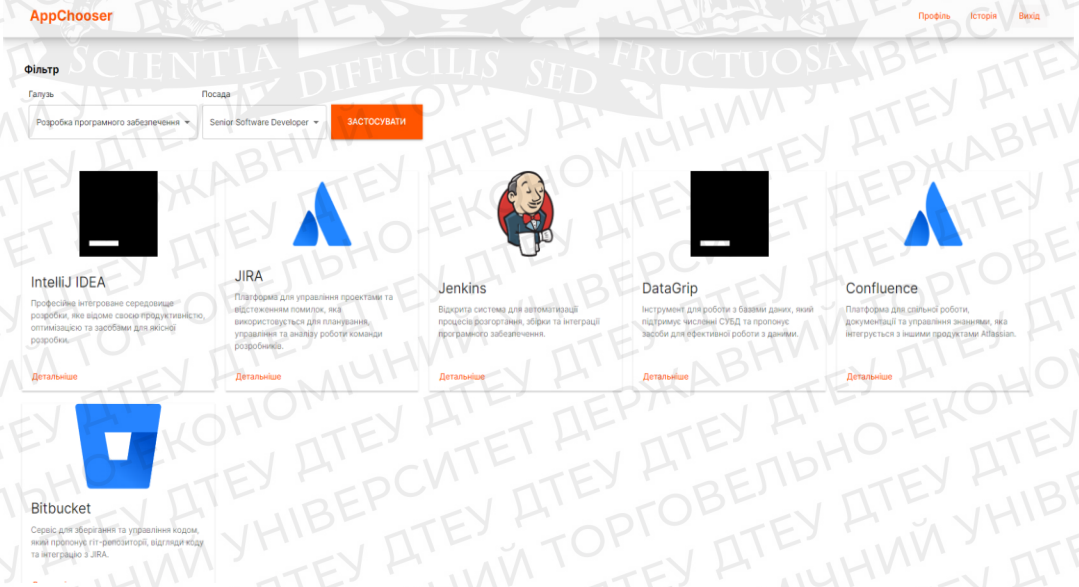

Рис 4.10. Основна сторінка з рекомендованим ПЗ
Джерело: розроблено автором

На даній сторінці розташовується рекомендоване програмне забезпечення.

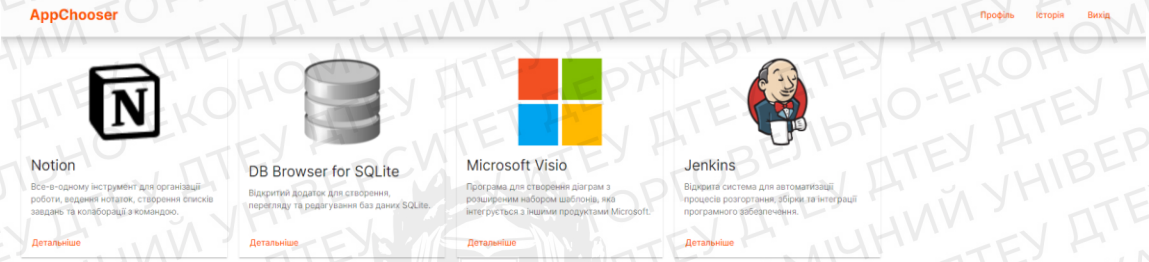

Рис. 4.11. Сторінка історії користувача

На даній сторінці розташована інформація про історію рекомендованого програмного забезпечення користовачу.

4.5. Висновок до розділу 4

В цьому розділі була описана структура програмного забезпечення, а саме Frontend- клієнтська частина застосунку та Backend. Були висвітлені та описані методи та основні функціональні частини програмного забезпечення.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Було проведено детальний аналіз та дослідження потреб користувача з метою визначення ключових критеріїв та параметрів для проекту. Розуміння та визначення потреб користувача є важливим етапом у розробці будь-якої інформаційної системи, оскільки воно дозволяє ефективно відповідати на запити ІТ-фахівців та задовольняти їхні конкретні вимоги. Проведено аналіз великих та відомих організацій та платформ у сфері ІТ, що дозволило вивчити основні тенденції ринку, а також визначити принципи та методи, які ці організації використовують для підбору та рекомендації ПЗ. Ця інформація є цінною для подальшого формування критеріїв підбору програмного забезпечення в рамках розроблюваної системи. Окрема увага приділена стандартам якості ПЗ. Сучасний ринок програмного забезпечення характеризується великою конкуренцією, тому важливо не лише вміти підібрати потрібне ПЗ, але й забезпечити, що воно відповідає всім необхідним стандартам якості. Врахування цих стандартів у процесі підбору дозволить ІТ-фахівцям бути впевненими у надійності та ефективності обраного рішення.

Також увага була приділена інструментам розробки, які є ключовими для успішного створення та реалізації професійно-орієнтованої експертної системи підбору програмного забезпечення.

Було підкреслено важливість вибору правильних інструментів для розробки. Правильний вибір інструментів впливає на продуктивність розробки, ефективність тестування, легкість підтримки та швидкість запуску

продукту. Як основний редактор коду для розробки було обрано IDE "Visual Studio Code". Цей інструмент відзначається високою гнучкістю, підтримкою великої кількості розширень та можливістю інтеграції з різними платформами і мовами програмування.Для роботи з базою даних обрано "MySQL Workbench", який є ефективним інструментом для проектування, розробки та адміністрування баз даних. Самою базою даних для системи обрано "MySQL", відому своєю надійністю, швидкістю та відкритістю.Для автентифікації користувачів та забезпечення безпеки системи використано "Google OAuth". Це дає можливість швидко та безпечно авторизовувати користувачів, використовуючи їхні облікові записи Google. Важливим етапом розробки стало створення "User Flow" – схеми взаємодії користувача з системою. Це допомогло розуміти основні точки контакту користувача та ефективно планувати реалізацію функціоналу.

Підсумовуючи, можна сказати, що обраний набір інструментів відіграє ключову роль у розробці системи. Кожен інструмент було обрано з метою оптимізації процесів розробки, забезпечення високої продуктивності та відповідності сучасним стандартам та вимогам до програмного забезпечення.

Була представлена загальна характеристика інформаційного забезпечення, в якій наголошено на ключових аспектах зберігання, обробки та передачі інформації в рамках системи. Розглянуто принципи зберігання інформації, її структуризації та доступу до неї.

Щодо типу носія даних, було визначено, що основним носієм для зберігання інформації в системі є база даних на основі MySQL. Даний вибір обумовлений надійністю, високою продуктивністю та широкими можливостями цієї системи управління базами даних.

Обґрунтування вибору технологій було зосереджено на аналізі потреб системи, її функціональності та вимог до продуктивності та надійності. Враховуючи специфіку задачі та особливості роботи IT-фахівців, було

вирішено використовувати саме ті технології, які найкраще відповідають цим критеріям.

Структура баз даних та інформаційних масивів була детально розглянута і представлена у вигляді схем та. Це допомогло визначити основні зв'язки між елементами даних, а також оптимізувати процеси зберігання та доступу до інформації.

Підсумовуючи, можна зазначити, що правильно організоване інформаційне забезпечення є одним з ключових аспектів успішної реалізації професійно-орієнтованої експертної системи підбору ПЗ. Вибір надійних технологій, а також ефективна структура бази даних, забезпечують високу продуктивність та зручність використання системи для ІТ-фахівців.

Структура клієнтської частини була розглянута з урахуванням зручності використання, інтуїтивності та забезпеченням ефективної взаємодії користувача з системою. Було виявлено ключові компоненти та їхні зв'язки, що сприяє підвищенню продуктивності користувача.

Інтерфейс користувача було представлено як ключовий елемент, що забезпечує ефективну взаємодію користувача з системою. Основний акцент було зроблено на зручності, інтуїтивності та адаптивності інтерфейсу для різних категорій користувачів.

Щодо пропозицій, штучний інтелект який дуже швидко розвивається в наші часи надзвичайно сильно міг би допомогти майже в кожній частині даного проекту. На його базі можна побудувати безліч алгоритмів аналізу програмного забезпечення для рекомендацій користувачам. Це збільшить кількість ПЗ та його різноманіття. Єдиної проблемою може стати імплементація ШІ в даний проект, так як при початковому проектуванні такий варіант не розглядався.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stack Overflow Developer Survey 2023 Електронний ресурс. URL: https://survey.stackoverflow.co/2023/#technology

2. GitHub's State of the Octoverse 2022 Електронний ресурс. URL: https://octoverse.github.com/2022/top-programming-languages

3. World Quality Report by Capgemini 2022-2023. Електронний ресурс. URL: https://www.capgemini.com/wp-content/uploads/2022/10/WQR-2022-Report-Final.pdf

4. Модель якості систем та програмного забезпечення iso/iec 25010 Електронний ресурс URL: https://cdn.standards.iteh.ai/samples/35733/2ca18b477b7845a5b8cae39d6de0c098/ISO-IEC-25010-2011.pdf

5. Google OAuth система автентифікації. Електронний ресурс. URL: https://developers.google.com/identity/protocols/oauth2

6. MySQL Workbenc інструмент розробки та адміністрування баз даних Електронний ресурс. URL: https://dev.mysql.com/doc/relnotes/mysql/8.0/en/news-8-0-0.html

7. Хмарні сервіси AWS Електронний ресурс. URL: https://aws.amazon.com/ru/

8. CMMI (Capability Maturity Model Integration) Електронний ресурс. URL: https://blogs.bmc.com/cmmi-capability-maturity-model-integration/?print-posts=pdf

9. AWS Elastic Beanstalk Розгортання та масштабування веб-додатків. URL: https://aws.amazon.com/ru/elasticbeanstalk/

					ДТЕУ 121 02-13.МР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		24.02.23	*Професійно-орієнтована експертна система підбору програмного забезпечення за запитами IT-фахівців*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Тищенко Д.О.		24.02.23		*СВД*	*62*	*62*
Гарант		Котенко Н.О.		24.02.23		*Факультет інформаційних технологій 2м курс, 2 група*		
Розробив		Осадчук М.Я		24.02.23	*Список використаних джерел*			

ТЕХНІЧНЕ ЗАВДАННЯ

Загальні положення:

Назва проекту: Професійно-орієнтована експертна система підбору ПЗ.

Мета проекту: Створення системи, що автоматизує процес вибору програмного забезпечення за специфічними запитами IT-спеціалістів.

Основні функції системи:

Отримання та аналіз запиту від користувача.

Визначення критеріїв для підбору програмного забезпечення.

Пошук в базі даних відповідного програмного забезпечення, що відповідає заданим критеріям.

Виведення результатів пошуку користувачу у зручному форматі.

Можливість збереження історії запитів користувача.

Технічні вимоги:

Мова програмування TypeScript, JavaScript.

Система управління базами даних: MySQL.

Інтерфейс: Веб-інтерфейс з адаптивним дизайном.

Інтеграція з зовнішніми ресурсами.

Інші вимоги:

Можливість масштабування системи.

Резервне копіювання бази даних.

Етапи реалізації:

Аналіз вимог та формулювання задач.

					ДТЕУ 121 02-13.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		15.03.23	*Професійно-орієнтована експертна система підбору програмного забезпечення за запитами IT-фахівців*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Тищенко Д.О.		15.03.23		*ТЗ*	*63*	*62*
Гарант		Котенко Н.О.		15.03.23		*Факультет інформаційних технологій 2м курс, 2 група*		
Розробив		Осадчук М.Я		15.03.23	*Технічне завдання*			

Проектування системи (включаючи базу даних, архітектуру програми, інтерфейс користувача).

Розробка прототипу системи.

Валідація системи. Фінальна реалізація системи та її впровадження.

ДОДАТКИ

Додаток А

```
import axios from 'axios';
import { toast } from 'react-toastify';

axios.defaults.withCredentials = true;

export const get = async <T>(path: string, body?: any) => {
 try {
  const { data } = await axios.get<T>(path, { ...body });

  return data;
 } catch (error: any) {
  if (error.response.data.statusCode !== 401) {
   toast.error(error.response.data.message);
  }
 }
};

export const post = async <T>(path: string, body?: any) => {
 try {
  const { data } = await axios.post<T>(path, { ...body });

  return data;
 } catch (error: any) {
  if (error.response.data.statusCode !== 401) {
   toast.error(error.response.data.message);
  }
 }
};

export const remove = async <T>(path: string, body?: any) => {
 try {
  const { data } = await axios.delete<T>(path, { ...body });

  return data;
 } catch (error: any) {
  if (error.response.data.statusCode !== 401) {
   toast.error(error.response.data.message);
  }
 }
};
```

```
export const patch = async <T>(path: string, body?: any) => {
  try {
    const { data } = await axios.patch<T>(path, { ...body });

    return data;
  } catch (error: any) {
    if (error.response.data.statusCode !== 401) {
      toast.error(error.response.data.message);
    }
  }
};
```

```
import { AreaType } from "../types/Area";
import { Profile, ProfileSave } from "../types/User";
import { ApplicationGetType, ApplicationType, HistoryType } from "../types/Application";
import { get, patch, post, remove } from "./fetch";

const API_URL = process.env.REACT_APP_API_URL;

export const login = (email: string) => {
 return post(`${API_URL}/auth/login`, { email });
}

export const checkOtp = (email: string, otp: number) => {
 return post(`${API_URL}/auth/check-otp`, { email, otp });
}

export const checkAuth = () => {
 return get<{ access_token: string }>(`${API_URL}/auth/token`);
}

export const logout = () => {
 return get(`${API_URL}/auth/logout`);
}

export const fetchUserInfo = (id: string) => {
 return get(`${API_URL}/airtable/user/${id}`);
}
```

```
export const profilesave = (profile: ProfileSave) => {
  return post(`${API_URL}/user/profile`, profile);
}

export const getareaposition = () => {
  return get<AreaType[]>(`${API_URL}/user/area-position`);
}

export const getapps = (getapps: ApplicationGetType) => {
  return post<ApplicationType[]>(`${API_URL}/user/apps`, getapps);
}

export const gethistory = () => {
  return get<HistoryType[]>(`${API_URL}/user/history`);
}

export const addhistory = (applicationId: number) => {
  return post(`${API_URL}/user/history`, { applicationId });
}
```

Додаток В

```
<?php

$link = mysqli_connect('localhost','root','root','electro');
if (mysqli_connect_errno())
{
	echo 'Ошибка ('.mysqli_connect_errno().'): '.mysqli_connect_error();
	exit();
}

?>
```

```
import {
  Button,
  CardContent,
  Divider,
  Link,
  Typography,
  styled,
} from "@mui/material";
import { Box } from "@mui/system";
import { Dispatch, FC, SetStateAction, useState } from "react";
import { useNavigate, useSearchParams } from "react-router-dom";
import { GoogleIcon } from "../icons/GoogleIcon";
import { InputError } from "../InputError";
import { validateEmail } from "../../utils/helpers";
import { AuthInputStyled } from "./AuthInputStyled";
import { login } from "../../api/requests";
import { LoadingButton } from "@mui/lab";
import { CardContainerStyled } from "./CardConatinerStyled";
import { toast } from 'react-toastify';

const DividerStyled = styled(Divider)`
&::before,
&::after {
  border-top: thin dashed rgb(0, 0, 0);
}
`;

interface SignUpProps {
```

```
  email: string;
  setEmail: Dispatch<SetStateAction<string>>;
}

export const SignUpForm: FC<SignUpProps> = ({ email, setEmail}) => {
  const [isError, setIsError] = useState(false);
  const [isLoading, setIsLoading] = useState(false);

  const [, setSearchParams] = useSearchParams();
  const navigate = useNavigate();

  const handleContinue = async () => {
   if (!validateEmail(email)) {
    setIsError(true);
    return;
   }

   try {
    setIsLoading(true);
    const response = await login(email);

    if (response) {
     setSearchParams({ verification: "true" });
    }
   } catch (err) {
    toast.error('Error in processing sign up');
   } finally {
    setIsLoading(false);
   }
  };
```

```
return (
 <CardContainerStyled>
  <CardContent
   sx={{
    display: "flex",
    flexDirection: "column",
    gap: "28px",
   }}
  >
   <Typography
    sx={{
     fontSize: "24px",
     fontWeight: 700,
    }}
   >
    Welcome to AppChooser
   </Typography>

   <Box
    sx={{
     display: "flex",
     flexDirection: "column",
     gap: "8px",
    }}
   >
    <Typography
     sx={{
      fontSize: "14px",
     }}
```

```
>
  Email
</Typography>

<AuthInputStyled
  variant="outlined"
  fullWidth
  value={email}
  onChange={(e) => {
    setEmail(e.target.value);
    setIsError(false);
  }}
  autoComplete="off"
/>

{isError && <InputError errorText="Email is not valid" />}
</Box>

<LoadingButton
  variant="contained"
  fullWidth
  sx={{
    borderRadius: 0,
    backgroundColor: "#FF5500",
    height: "44px",
    ":hover": {
      backgroundColor: "#CD4400",
    },
    opacity: email.trim() ? 1 : 0.6,
    pointerEvents: email.trim() ? "all" : "none",
```

```
        }}
        onClick={handleContinue}
        loading={isLoading}
      >
        Continue
      </LoadingButton>

      <DividerStyled>
        OR
      </DividerStyled>

      <Link                                    href="http://appchooser.us-east-
1.elasticbeanstalk.com/auth/google">
        <Button
          variant="outlined"
          fullWidth
          startIcon={<GoogleIcon />}
          sx={{
            borderColor: "#D9D9DA",
            color: "#070707",
            textTransform: "none",
            fontSize: "14px",
            height: "44px",
          }}
        >
          Join with Google
        </Button>
      </Link>
    </CardContent>
  </CardContainerStyled>
```

```
    );
};`;
```

```
import { ListItem, ListItemAvatar, ListItemText } from "@mui/material";
import { FeatureItemType } from "../types/FeatureItem";
import { FC } from "react";
import { CheckIcon } from "./icons/CheckIcon";

interface FeatureItemProps {
 feature: FeatureItemType;
}

export const FeatureItem: FC<FeatureItemProps> = ({ feature }) => (
 <ListItem
  alignItems="flex-start"
  sx={{
   gap: '10px',
   padding: 0
  }}
 >
  <ListItemAvatar
   sx={{
    minWidth: '0',
    width: 'min-content'
   }}
  >
   <CheckIcon />
  </ListItemAvatar>

  <ListItemText
   primary={feature.title}
```

```
      secondary={feature.text}
      primaryTypographyProps={{
        fontSize: '16px',
        fontWeight: 500
      }}
      secondaryTypographyProps={{
        fontSize: '14px',
        color: '#464646'
      }}
    />
  </ListItem>
);  );
}; `;
```

```
import { Box, Button, Link, Typography, styled } from "@mui/material";
import logoHeader from '../images/logo-chat.png';
import titleHeader from '../images/header-title.png';
import { DownoadIcon } from "./icons/DownloadIcon";
import { Link as LinkDom, useNavigate, useParams } from "react-router-dom";
import { logout } from "../api/requests";
import { useAppDispatch, useAppSelector } from "../redux/hooks";
import { setAuth } from "../redux/authSlice";
import { toast } from "react-toastify";
import { Dispatch, FC, useState, SetStateAction, useEffect } from "react";

const HeaderContainerStyled = styled(Box)`
  position: relative;
  z-index: 5;
  padding: 16px 48px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  background-color: #FCFCFC;
  border-bottom: 1px solid #DCDCDC;
  box-shadow: 0px 0px 0px 0px rgba(0, 0, 0, 0.10), 0px 1px 3px 0px rgba(0, 0, 0, 0.10), 0px 5px 5px 0px rgba(0, 0, 0, 0.09), 0px 12px 7px 0px rgba(0, 0, 0, 0.05), 0px 21px 8px 0px rgba(0, 0, 0, 0.01), 0px 33px 9px 0px rgba(0, 0, 0, 0.00);
`;

interface HeaderProps {
  setAppLoading: Dispatch<SetStateAction<boolean>>;
```

```
}

export const Header: FC<HeaderProps> = ({ setAppLoading }) => {
  const [pdfConfigs, setPdfConfigs] = useState({});
  const [img, setImg] = useState('');
  const { userId } = useAppSelector(state => state.auth);

  const navigate = useNavigate();

  const dispatch = useAppDispatch();
  const { id } = useParams();

  useEffect(() => {
    const getConfigs = async () => {
      if (id) {

      }
    }

    getConfigs();
  }, [id]);

  const handleLogout = async () => {
    try {
      await logout();
      dispatch(setAuth(null));
      navigate('/signup');
    } catch {
      toast.error('Error while processing log out. Please, try again.')
    }
```

```
}

return (
 <HeaderContainerStyled>
  <LinkDom style={{
   textDecoration: 'none',
  }} to='/'>
   <Typography
    sx={{
     fontSize: "24px",
     fontWeight: 700,
     color: "#FF5500",
    }}
    >
    AppChooser
   </Typography>
  </LinkDom>

  <Box
   sx={{
    display: 'flex',
    gap: '16px'
   }}
  >
   {userId && (
    <Link href='/profile'>
    <Button
     sx={{
      color: '#FF5500',
      textTransform: 'none',
```

```
      fontSize: '14px',
      paddingX: '10px',
      borderRadius: 0,
      ":hover": {
       backgroundColor: "#F7F6F6",
      },
     }}
    >
     Профіль
    </Button>
   </Link>
  )}

  {userId && (
   <Link href='/history'>
   <Button
    sx={{
     color: '#FF5500',
     textTransform: 'none',
     fontSize: '14px',
     paddingX: '10px',
     borderRadius: 0,
     ":hover": {
      backgroundColor: "#F7F6F6",
     },
    }}
   >
    Історія
   </Button>
  </Link>
```

```
        )}

        {userId && (
          <Button
            sx={{
              color: '#FF5500',
              textTransform: 'none',
              fontSize: '14px',
              paddingX: '10px',
              borderRadius: 0,
              ":hover": {
                backgroundColor: "#F7F6F6",
              },
            }}
            onClick={handleLogout}
          >
            Вихід
          </Button>
        )}
      </Box>
    </HeaderContainerStyled>
  );
}
```

```
<?php
 require_once 'database.php';
 require_once 'function.php';
 ?>
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/css/bootstrap.min.css"
rel="stylesheet"                                                integrity="sha384-
KyZXEAg3QhqLMpG8r+8fhAXLRk2vvoC2f3B09zVXn8CA5QIVfZOJ3BCsw2
P0p/We" crossorigin="anonymous">
 <?php include('css.php') ?>
 <title>NKlaus</title>
</head>
<body>
 <?php include('header.php') ?>
 <main>
        <div class="row m-0">
            <div class="col-12 col-md-4 p-0">
                <?php include('aside.php') ?>
            </div>
            <div class="main_section col-12 col-md-8">
                <div class="col-12 d-flex flex-column align-items-
center">
                    <h1>Обране</h1>
```

```
                </div>
                <div class="row">
                    <?php
                        $categories = get_categories($link,"wish");
                     ?>
                    <?php foreach ($categories as $wish): ?>
                    <div class="col-lg-4 col-6 p-2">
                        <div class="wrapper col-12 p-2 d-flex flex-column align-items-center justify-content-between">
                            <img <?=$wish["fotoUrl"] ?> alt="comp" class="img-fluid">
                            <h4><?=$wish["name"] ?></h4>
                            <p><?=$wish["description"] ?></p>
                            <div class="price my-2">Ціна: <span><?=$wish["price"] ?></span></div>
                        </div>
                    </div>
                    <?php endforeach; ?>
                </div>
            </div>
        </div>
    </main>
    <?php include('footer.php') ?>


    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/js/bootstrap.bundle.min.js" integrity="sha384-U1DAWAznBHeqEIlVSCgzq+c9gqGAJn5c/t99JyeKa9xxaYpSvHU5awsuZVVFIhvj" crossorigin="anonymous"></script>
```

```
</body>
</html>
```

```
import React, { useEffect, useState } from "react";
import { Outlet, useNavigate } from "react-router-dom";
import { useAuth } from "./hooks/useAuth";
import { ToastContainer } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';

import { Box } from "@mui/material";
import { Header } from "./components/Header";
import { useAppDispatch, useAppSelector } from "./redux/hooks";
import { fetchUserInfo } from "./api/requests";
import { Loader } from "./components/Loader";

export const App: React.FC = () => {
 const { isAuthLoading } = useAuth();
 const [isLoading, setIsLoading] = useState(false);
 const { userId } = useAppSelector(state => state.auth);
 const dispatch = useAppDispatch();
 const navigate = useNavigate();

 useEffect(() => {
  if (userId) {
    const getUserInfo = async () => {
     const info: any = await fetchUserInfo(userId);

     const regDaysTrial = (Date.now() - Number(info.created_at)) / (24 * 60 * 60 *
1000) - 7;
     if (regDaysTrial > 0) {
      return;
     }
    }

    getUserInfo();
  }
 }, [dispatch, navigate, userId]);

 if (isAuthLoading || isLoading) {
  return <Loader />
 }

 return (
  <Box>
   <ToastContainer
```

```
        position="bottom-right"
        autoClose={3000}
        limit={8}
        newestOnTop
        progressStyle={{ background: '#FF5500' }}
      />

      <Header setAppLoading={setIsLoading} />

      <Outlet />
    </Box>
  );
};
```

```
import {
  Body,
  Controller,
  Get,
  Param,
  Post,
  Redirect,
  Req,
  Res,
  UseGuards,
} from '@nestjs/common';
import { ApiTags, ApiOperation, ApiResponse } from '@nestjs/swagger';
import { AuthService } from './auth.service';
import { LoginUserDto } from './dto/login.user.dto';
import { BadRequestException } from '@nestjs/common';
import { checkUserOtpDto } from './dto/check.otp.dto';
import { AuthGuard } from '@nestjs/passport';
import { ConfigService } from '@nestjs/config';
import { Request, Response } from 'express';
import { JwtAuthGuard } from './guards/auth.guard';

@ApiTags('Auth')
@Controller('auth')
export class AuthController {
  googleRedirectUrl: string;
  constructor(
    private readonly authService: AuthService,
    private readonly configService: ConfigService,
  ) {
    this.googleRedirectUrl                                                    =
this.configService.get('GOOGLE_FRONT_REDIRECT');
  }

  @Post('login')
  @ApiOperation({ summary: 'User sign in and sign up' })
  @ApiResponse({
    status: 201,
  })
  async login(@Body() body: LoginUserDto) {
    try {
      const response = await this.authService.login(body);
      return {
```

```
      response,
    };
  } catch (error) {
    throw new BadRequestException(error.message);
  }
}

@Post('check-otp')
@ApiOperation({ summary: 'Check user OTP code' })
@ApiResponse({
  status: 201,
})
async checkOTP(
  @Body() body: checkUserOtpDto,
  @Res({ passthrough: true }) res,
) {
  try {
    const token = await this.authService.checkOTP(body);
    if (!('access_token' in token)) {
      throw new BadRequestException('Wrong OTP', 'Wrong OTP');
    }

    res.cookie('access_token', token.access_token, {
      maxAge: 168 * 60 * 60 * 1000,
      httpOnly: true,
    });
    return token.access_token;
  } catch (error) {
    throw new BadRequestException(error.message, error.message);
  }
}

@ApiOperation({
  summary: 'Redirect to google auth',
  description: 'Use this route to redirect to google auth',
})
@UseGuards(AuthGuard('google'))
@Get('google')
googleAuth(@Res({ passthrough: true }) res) {}

@ApiOperation({
  summary: 'Google auth callback',
  description: 'This route is only for google service',
})
```

```
@UseGuards(AuthGuard('google'))
@Redirect()
@Get('google/redirect')
async googleAuthRedirect(@Req() req, @Res({ passthrough: true }) res) {
  const response = await this.authService.googleLogin(req.user);
  res.cookie('access_token', response.access_token, {
    maxAge: 168 * 60 * 60 * 1000,
    httpOnly: true,
  });
  return { url: this.googleRedirectUrl };
}

@Get('token')
@ApiOperation({ summary: 'User token check' })
@UseGuards(JwtAuthGuard)
@ApiResponse({
  status: 200,
})
checkToken(@Req() request: Request) {
  try {
    const cookies = request.headers.cookie.split('; ');
    const tokenCookie = cookies.find((cookie) =>
      cookie.startsWith('access_token='),
    );

    if (tokenCookie) {
      return { access_token: tokenCookie.split('=')[1] };
    }

    return null;
  } catch (error) {
    throw new BadRequestException(error.message);
  }
}

@Get('/logout')
@UseGuards(JwtAuthGuard)
async logOut(@Res() res: Response) {
  try {
    res.clearCookie('access_token');
    res.send('Logout success');
    return { message: 'logout success' };
  } catch (error) {
    throw new BadRequestException(error.message);
```

```
    }
  }
}
```

```
import { Injectable } from '@nestjs/common';
import { LoginUserDto } from './dto/login.user.dto';
import { UserEntity } from 'src/user/entity/user.entity';
import { generateRandomNumber } from 'src/helpers/function';
import { MailService } from 'src/mail/mail.service';
import { BadRequestException } from '@nestjs/common';
import { checkUserOtpDto } from './dto/check.otp.dto';
import { TokenService } from 'src/token/token.service';
import { GoogleAuthUser } from './interfaces/google.auth.user.interface';
import { Repository } from 'typeorm';
import { InjectRepository } from '@nestjs/typeorm';
@Injectable()
export class AuthService {
 constructor(
  @InjectRepository(UserEntity)
  private readonly userRepository: Repository<UserEntity>,
  private readonly mailService: MailService,
  private readonly tokenService: TokenService,
 ) {}

 async login({ email }: LoginUserDto) {
  try {
   const _user = await this.userRepository.findOne({ where: { email } });
   const otp = generateRandomNumber();
   if (!_user) {
    await this.userRepository.save({
     email,
     otp,
    });

    await this.mailService.sendOTPEmail(email, otp);
    return 'Email has been sended';
   }

   await this.userRepository.update(_user.id, { otp });
   await this.mailService.sendOTPEmail(email, otp);
   return 'Email has been sended';
  } catch (error) {
   console.log(error);

   throw new BadRequestException(error.message, error.message);
  }
```

```
}

async googleLogin(userInfo: GoogleAuthUser) {
  const _user = await this.userRepository.findOne({
    where: { email: userInfo.email },
    relations: ['profile', 'profile.area', 'profile.position']
  });
  if (!_user) {
    const newCustomer = await this.userRepository.save({
      email: userInfo.email,
    });
    return {
      access_token: this.tokenService.generateAccessToken({
        id: newCustomer.id,
        email: newCustomer.email,
      }),
    };
  } else {
    return {
      access_token: this.tokenService.generateAccessToken({
        id: _user.id,
        email: _user.email,
        profile: {
          companyName: _user.profile?.companyName,
          area: _user.profile?.area,
          position: _user.profile?.position,
          techStack: _user.profile?.techStack.split(',') as string[],
          experience: _user.profile?.experience,
        }
      }),
    };
  }
}

async checkOTP({ email, otp }: checkUserOtpDto) {
  try {
    const _user = await this.userRepository.findOne({
      where: { email },
      relations: ['profile', 'profile.area', 'profile.position']
    });

    if (!_user)
      return new BadRequestException(
        'User is not found',
```

```
        'User is not found',
      );

    if (_user.otp !== otp)
      return new BadRequestException('OTP wrong', 'OTP wrong');

    await this.userRepository.update(_user.id, { otp: null });

    return {
      access_token: this.tokenService.generateAccessToken({
        id: _user.id,
        email: _user.email,
        profile: {
          companyName: _user.profile?.companyName,
          area: _user.profile?.area,
          position: _user.profile?.position,
          techStack: _user.profile?.techStack.split(',') as string[],
          experience: _user.profile?.experience,
        }
      }),
    };
  } catch (error) {
    throw new BadRequestException(error.message, error.message);
  }
 }
}
```

```
import { Controller, Get, Post, Req, Res, Body, Request } from '@nestjs/common';
import { ApiOperation, ApiResponse, ApiTags } from '@nestjs/swagger';
import { UseGuards } from '@nestjs/common';
import { JwtAuthGuard } from 'src/auth/guards/auth.guard';
import { UserService } from './user.service';
import { saveProfile } from './dto/save.profile.dto';
import { GetAppsDto } from './dto/get.apps.dto';
import { AddHistoryDto } from './dto/add.history.dto';

@ApiTags('User')
@Controller('user')
export class UserController {
 constructor(private readonly userService: UserService){}

 @UseGuards(JwtAuthGuard)
 @ApiOperation({ summary: 'Save user profile' })
 @ApiResponse({
  status: 200,
 })
 @Post('profile')
 async decreaseCredit(
  @Body() body: saveProfile,
  @Request() req: any,
  @Res({ passthrough: true }) res,
 ){
    const result = await this.userService.saveprofile(req.user, body);
    res.cookie('access_token', result.access_token, {
     maxAge: 168 * 60 * 60 * 1000,
     httpOnly: true,
    });
    return result;
 }

 // @UseGuards(JwtAuthGuard)
 @ApiOperation({ summary: 'get arees and positions' })
 @ApiResponse({
  status: 200,
 })
 @Get('area-position')
 async getareaposition(
  @Request() req: any,
 ){
```

```
    const result = await this.userService.getareaposition();
    return result;
  }

  @UseGuards(JwtAuthGuard)
  @ApiOperation({ summary: 'get apps by filter' })
  @ApiResponse({
   status: 200,
  })
  @Post('apps')
  async getapps(
   @Body() body: GetAppsDto,
   @Request() req: any,
  ){
    const result = await this.userService.getapps(body.areaId, body.positionId);
    return result;
  }

  @UseGuards(JwtAuthGuard)
  @ApiOperation({ summary: 'add history' })
  @ApiResponse({
   status: 200,
  })
  @Post('history')
  async addhistory(
   @Body() body: AddHistoryDto,
   @Request() req: any,
  ){
    const result = await this.userService.addhistory(body.applicationId, req.user);
    return result;
  }
  @UseGuards(JwtAuthGuard)
  @ApiOperation({ summary: 'get history' })
  @ApiResponse({
   status: 200,
  })
  @Get('history')
  async gethistory(
   @Request() req: any,
  ){
    const result = await this.userService.gethistory(req.user);
    return result;
  }
}
```

```
import { BadRequestException, Injectable } from '@nestjs/common';
import { UserEntity } from './entity/user.entity';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository, Unique } from 'typeorm';
import { saveProfile } from './dto/save.profile.dto';
import { ProfileEntity } from './entity/profile.entity';
import { TokenService } from 'src/token/token.service';
import { AreaEntity } from './entity/area.entity';
import { PositionEntity } from './entity/position.entity';
import { HistoryEntity, HistoryResultEntity } from './entity/history.entity';
import { ApplicationEntity } from './entity/application.entity';

@Injectable()
export class UserService {
 constructor(
  @InjectRepository(UserEntity)
  private readonly userRepository: Repository<UserEntity>,

  @InjectRepository(AreaEntity)
  private readonly areaRepository: Repository<AreaEntity>,

  @InjectRepository(PositionEntity)
  private readonly positionRepository: Repository<PositionEntity>,

  @InjectRepository(HistoryEntity)
  private readonly historyRepository: Repository<HistoryEntity>,

  private readonly tokenService: TokenService,
 ) {}

 async saveprofile(id: number, profile: saveProfile) {
  const user = await this.userRepository.findOne({
   where: { id },
   relations: ['profile', 'profile.area', 'profile.position']
  });
  if (!user) throw new BadRequestException("User is not found");

  console.log(profile);

  const area = await this.areaRepository.findOne({ where: { id: profile.areaId } });
  const position = await this.positionRepository.findOne({ where: { id:
profile.positionId } });
```

```
  if(!user.profile) {
    user.profile = {
      companyName: profile.companyName,
      area: area,
      position: position,
      techStack: profile.techStack.join(','),
      experience: profile.experience,
    } as ProfileEntity;

    await this.userRepository.save(user);
  }
  else{
    user.profile.companyName = profile.companyName;
    user.profile.area = area;
    user.profile.position = position;
    user.profile.techStack = profile.techStack.join(',');
    user.profile.experience = profile.experience;

    await this.userRepository.save(user);
  }

  return {
    access_token: this.tokenService.generateAccessToken({
      id: user.id,
      email: user.email,
      profile: {
        companyName: user.profile.companyName,
        area: user.profile.area,
        position: user.profile.position,
        techStack: user.profile.techStack.split(',') as string[],
        experience: user.profile.experience,
      }
    }),
  };
}

async getareaposition() {
  return await this.areaRepository.find({relations: ['position']});
}

async getapps(areaId: number, positionId: number) {
  const apps = await this.areaRepository.createQueryBuilder("area")
  .innerJoinAndSelect("area.position", "position")
```

```
    .innerJoinAndSelect("position.application", "application")
    .where("position.id = :positionId AND area.id = :areaId", { positionId, areaId })
    .getMany();

   return apps[0]?.position[0]?.application;
  }

  async addhistory(applicationId: number, userId: number) {
   const res = await this.historyRepository.save({
    applicationId: applicationId,
    userId: userId
   });

   return res;
  }

  async gethistory(userId: number) {
   const res = await this.historyRepository.createQueryBuilder("history")
    .leftJoinAndMapMany("history.application",  ApplicationEntity,  "application",
"history.applicationId = application.id")
    .where("history.userId = :userId", { userId })
    .orderBy("history.id", "DESC")
    .getMany();
   console.log(res);

   return res;
  }
}
```

```
import { Logger, Module, OnModuleInit } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { ConfigModule, ConfigService } from '@nestjs/config';
import * as path from 'path';
import { ServeStaticModule } from '@nestjs/serve-static';
import { TypeOrmModule } from '@nestjs/typeorm';
import { AuthModule } from './auth/auth.module';
import { UserModule } from './user/user.module';
import { MailModule } from './mail/mail.module';
import { PassportModule } from '@nestjs/passport';
import { UserEntity } from './user/entity/user.entity';
import { ProfileEntity } from './user/entity/profile.entity';
import { AreaEntity } from './user/entity/area.entity';
import { PositionEntity } from './user/entity/position.entity';
import { ApplicationEntity } from './user/entity/application.entity';
import { HistoryEntity } from './user/entity/history.entity';

@Module({
 imports: [
  PassportModule.register({ defaultStrategy: 'google' }),
  ConfigModule.forRoot(),
  ServeStaticModule.forRoot({
   rootPath: path.join(__dirname, '..', 'client', 'build'),
  }),
  TypeOrmModule.forRoot({
   type: 'mysql',
   host: 'lproject-1.cubb3ge37ocp.us-east-1.rds.amazonaws.com',
   port: 3306,
   username: 'lp_admin',
   password: 'Mobqac6k',
   database: 'lproject-1',
   entities: [UserEntity, ProfileEntity, AreaEntity, PositionEntity,
ApplicationEntity, HistoryEntity],
   synchronize: true,
  }),
  AuthModule,
  UserModule,
  MailModule,
 ],
 controllers: [AppController],
 providers: [AppService],
```

```
})
export class AppModule implements OnModuleInit {
 constructor() {}

 async onModuleInit() {
 }
}
```