

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ БІРЖІ ФРІЛАНСУ
FREELANCE PLATFORM SOFTWARE

Виконав: студент(ка) 4 курсу, групи КНТ-138
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Денисенко С.В.

(прізвище та ініціали)

Керівник Льовкін В.М

(прізвище та ініціали)

Рецензент Скрупський С.Ю.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
« » 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Денисенка Сергія Володимировича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення біржі фрілансу
Freelance Platform Software

керівник проєкту (роботи) Льовкін Валерій Миколайович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «27» квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) червень 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.
2. Проєктування програми. 3 Розробка програми. 4. Експлуатація та тестування
програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Льовкін В.М., доцент		
Нормоконтроль	Камінська Ж.К., асистент		

7. Дата видачі завдання «04» квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Проектування програми	3 тиждень	Розділ 2
4	Розробка програми	4-5 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
7	Захист роботи	8 тиждень	

Студент(ка)

_____ Денисенко С.В.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи)

_____ Льовкін В.М.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
144 с., 96 рис., 2 табл., 3 дод., 36 джерел.

ВЕББІРЖА ФРІЛАНСИНГУ, WEBAPI, MICROSOFT SQL SERVER, IDENTITY, AUTOMAPER, MVC.

Об'єкт дослідження – процес встановлення взаємоз'язків між замовником та фрілансером.

Предмет дослідження – програмне забезпечення біржі фрілансу.

Мета роботи – розробити програмне забезпечення біржі фрілансу для підтримки взаємовідносин між замовниками та фрілансерами.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, мови програмування C#, платформа WebApi, система керування БД Microsoft SQL Server, багат шарова архітектура, генератор HTML сторінок Razor, Identity, AutoMapper, фреймворк MVC, Entity Framework Core, інтегроване середовище розробки Visual Studio, мови HTML і CSS, бібліотека Bootstrap.

Результати. Створено програмний застосунок, в основі серверну частину, яка підтримує авторизацію та реєстрацію користувачів, підтримує ретельні пошук фрілансерів та опис можливостей самого фрілансеру, в застосунку наявна можливість взаємодії користувачів за допомогою чату, а також є можливість швидкої зміни ролі, чого немає у аналогів.

Висновки. Проведено аналіз предметної області, визначено актуальність проблеми та проаналізовано аналоги. Розроблено застосунок, що надає функціональність для взаємодії найманого працівника (фрілансера) та роботодавця. Були розглянуті принципові архітектури, які використовуються на даний момент, та вибрана найбільш підходяща для даного застосунку.

Галузь використання – простір інтернет, може бути використано для налагодження середовища для взаємодії найманого працівника та роботодавця.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 144 pages, 96 figures, 2 tables, 3 appendixes, 36 sources.

WEB EXCHANGE FREELANCE, WEBAPI, MICROSOFT SQL SERVER, IDENTITY, AUTOMAPER, MVC.

The object of research is the process of establishing relationships between the customer and the freelancer.

The subject of research is freelance exchange software.

The purpose of the work is to develop freelance exchange software to maintain the relationship between customers and freelancers.

Materials, methods and technical means: structural and object-oriented programming, C # programming languages, WebApi platform, Microsoft SQL Server database management system, multilayer architecture, Razor HTML page generator, Identity, AutoMapper, MVC framework, Entity Framework Core, integrated Visual Studio development environment, HTML and CSS languages, Bootstrap library.

Results. Created a software application , based on the server part, which supports authorization and registration of users, supports thorough search for freelancers and describes the capabilities of the freelancer, the application has the ability to interact with users via chat, and can quickly change roles, which has no analogues.

Conclusions. The analysis of the subject area is carried out, the urgency of the problem is determined and the analogues are analyzed. An application has been developed that provides functionality for the interaction of an employee (freelancer) and an employer. The principal architectures currently in use have been considered and the most suitable for the application selected.

Field of use - the Internet space, can be used to create an environment for the interaction of employees and employers.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	9
Вступ.....	10
1 Аналіз предметної області	11
1.1 Основні поняття і визначення.....	11
1.2 Детальний опис об'єкта і предмета дослідження	13
1.2.1 Загальний огляд логіки появи фрілансової біржі	13
1.2.2 Актуальність фрілансової біржі	14
1.3. Кількісні та якісні показники об'єкта	16
1.3.1 Економічне середовище	20
1.3.2 Зрілість ставлення працівників до фрілансу	21
1.3.3 Правове середовище	22
1.4 Огляд існуючих аналогів.....	23
1.4.1 Фрілансова біржа «Upwork»	23
1.4.1.1 Опис.....	23
1.4.1.2 Переваги.....	23
1.4.1.3 Недоліки.....	24
1.4.2 Фрілансова біржа «Fiverr».....	25
1.4.2.1 Опис.....	25
1.4.2.2 Переваги.....	25
1.4.2.3 Недоліки.....	26
1.4.3 Фрілансова біржа «99designs»	27
1.4.3.2 Опис.....	27
1.4.3.3 Переваги.....	27
1.4.3.4 Недоліки.....	27
1.4.4 Фрілансова біржа «freelancer»	28
1.4.4.1 Визначення	28
1.4.4.2 Опис.....	28
1.4.4.3 Переваги.....	29
1.4.4.4. Недоліки.....	29
1.4.6 Фрілансова біржа «Guru»	30

	7
1.4.6.1 Визначення	30
1.4.6.2 Опис	30
1.4.6.3 Переваги	30
1.4.6.4. Недоліки	30
1.4.7 Фрілансова біржа «ServiceScape»	31
1.4.7.1 Визначення	31
1.4.7.2 Опис	31
1.4.7.3 Переваги	32
1.4.7.4. Недоліки	32
1.5 Порівняльна характеристика існуючих аналогів та розроблюваного програмного продукту	33
1.6 Висновки за першим розділом	35
2 Проєктування програми	37
2.1 Вибір мови програмування	37
2.2 Проєктування архітектури програмного забезпечення та огляд варіантів	40
2.2.1 Монолітна архітектура	40
2.2.2 Багатошарова архітектура	41
2.2.3 Onion-архітектура	42
2.2.4 Мікросервісна архітектура	44
2.2.5 Helix-архітектура	45
2.3 Функціональні вимоги до програмного забезпечення	46
2.4 Проєктування структури БД	50
2.5 Структура програмного проєкту	57
2.6 Вимоги до програмного забезпечення	60
2.7 Висновки за другим розділом	60
3 Розробка програми	61
3.1 Алгоритм функціонування програмного забезпечення	61
3.2 Опис модулів реалізованої програми	63
3.3 Висновки за розділом 3	78
4 Експлуатація та тестування програми	80

	8
4.1 Призначення програми	80
4.2 Умови виконання програми	80
4.3 Виконання програми	81
4.4 Тестування програми	92
Висновки	96
Перелік джерел посилання	98
Додаток А Технічне завдання	101
Додаток Б Текст програми	105
Додаток В Слайди презентації	137

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

BLL	– Business logic Layer;
DAL	– Data Access Layer;
DI	– Dependency Injection;
DTO	– Data Transfer Object
EF	– Entity Framework
HTTP	– Hyper Text Transfer Protocol;
IoC	– Inversion of Control;
IL	– Intermediate Language
Kiss	– Keep It Simple, Stupid;
MVC	– Model View Controller;
SOLID	– Single responsibility principle, Open-closed principle, Liskov substitution principle, Interface segregation principle, Dependency inversion principle;
SoC	– Separation of Concerns
TDD	– Test Driven Development;
URL	– Uniform Resource Locator;
UIL	– User Interface Layer
БД	– База Даних
ВВП	– Валовий внутрішній продукт

ВСТУП

Сучасне суспільство не стоїть на місці та шукає нових видів застосування своїх навичок та вмінь для забезпечення гідного рівня життя. Особливо у тяжкий кризовий час чи під час пандемії, коли йде скорочення робочих місць через неспроможність комерційних організацій, а роботодавці застосовують дедалі більші вимоги до працівників при малій заробітній платі, люди починають шукати самореалізацію за іншими формами діяльності.

У нашому суспільстві існує доволі багато соціальних проблем, які впливають на якість роботи. Існує проблема соціальної мобільності, коли людина прагне підійматися по кар'єрним сходам, проте це обмежується системою чинних зв'язків у колективі (споріднених чи знайомих).

Ще до недавнього часу було проблемою знайти собі роботу, бо людина була обмежена територією стартових можливостей, а люди, які приходили у нові сфери працевлаштування, проходили надскладний шлях. На сьогоднішній день ці проблеми намагаються вирішити за допомогою випробувальних термінів, курсів та іншого. Велика кількість інформації з'явилася з появою нових методів комунікації таких, як інтернет. Проте залишається ще багато проблем, пов'язаних із теперішньою системою трудового регулювання.

З розвитком сучасних технологій з'явилася досить нова сфера – фрілансова біржа. Це доволі нова сфера зайнятості людей, яка має чимало проблем, проте вона і вирішує чимало сучасних проблем. Так фрілансер (найманий працівник) стає вільною одиницею, що у своїй природі породжує конкуренцію. Це стимулює таких робітників до саморозвитку, що покращує послуги та свідомість таких людей у цілому.

Мета роботи було розроблення програмного забезпечення біржі фрілансу для підтримки взаємовідносин між замовниками та фрілансерами.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

В даному розділі проаналізовані основні проблеми та тематика, яка впливає на розвиток фрілансових бірж. Розглянуті поняття фрілансу як елемента економіки спільного споживання. У розділі розглянуто найбільш відомі фрілансові біржі у різних сегментах економіки та в різних країнах, їх переваги і недоліки.

1.1 Основні поняття і визначення

Біржа фріланса – онлайн платформа, що є зручним способом взаємодії між роботодавцем і спеціалістом. Даний формат співпраці не вимагає особистої зустрічі, а обговорювати всі деталі проєкту та оплачувати виконану працю можливо навіть тоді, коли сторони знаходяться в різних куточках планети [1].

Фрілансер – вільний працівник. Колись таким словом іменували вільних найманців у військовій справі. Сучасний фрілансер – це професіонал, який звільнився від офісного гніту і почав працювати вдома. Причому без обов'язкової офіційної освіти. Головне – досконало володіти навичками, необхідними для реалізації конкретного завдання [1].

Шерингова економіка – це нова культура та економічна бізнес-модель, коли за допомогою технологій та онлайн платформ люди можуть обмінюватися активами, які вони не використовують. Основний принцип – доступ до користування благом, а не володіння ним [2]-[3].

Нерегламентована зайнятість – незареєстрована органами державної влади і неоподаткована оплачувана діяльність. Це передусім робота без укладення трудових угод, комерційний туризм, надання послуг на бартерній основі та інше [4].

Самозайнятість – незалежна трудова діяльність, самостійне виробництво товарів і послуг без залучення найманої праці поза традиційними організаційними, професійними структурами та зовнішнім контролем [5].

Інтерфейс – сукупність методологій, засобів, шаблонів, методів і правил взаємодії між елементами системи і користувачем. Зазвичай він виступає як двонаправлений потік. Методами взаємодії з інтерфейсом у сфері операційних систем можуть виступати слайдери, кнопки, повзунки, стрілочки та інші елементи керування, які широко розповсюдженні.

Віддалена робота – форма зайнятості при якій найманий працівник працює вдома чи у місці, яке не передбачене для роботи замовником. Такий вид взаємозв'язку передбачує передачу інформації на відстані за допомогою сучасних засобів.

Каршеринги – короткострокова оренда чи прокат послуги або предмету.

Краунфандинг – використання невеликих сум капіталу від великої кількості людей для фінансування нового комерційного підприємства, він використовує легкий доступ до великих мереж людей через соціальні мережі та вебсайти краудфандингу, щоб об'єднати інвесторів та підприємців, з потенціалом збільшення підприємництва за рахунок розширення пулу інвесторів за межі традиційного кола власників, родичів та венчурних капіталістів [6].

Chief Operating Officer (COO) – операційний директор [7].

Chief Security Officer (CSO) – директор з безпеки [7].

Chief Executive Officer (CEO) – головний виконавчий директор [7].

Аутсорсинг – це ділова практика найму сторони поза компанією для надання послуг або створення товарів, які традиційно виконувались власними співробітниками та персоналом компанії. Або це практика, що зазвичай застосовується компаніями як міра зі скорочення витрат. Таким чином, це може вплинути на широкий спектр робочих місць від підтримки клієнтів до виробництва та бек-офісу [8].

1.2 Детальний опис об'єкта і предмета дослідження

1.2.1 Загальний огляд логіки появи фрілансової біржі

Сучасний світ розвивається прискореними темпами. Цьому сприяє поява нових методів комунікації. У процесі інтеграції інфокомунікаційного та фінансового ринків відбувається становлення нових форм фінансових послуг, реалізованих через різні комунікаційні канали. Найбільшу частину вкладу в це зробив інтернет як засіб для одночасної взаємодії з великою кількістю клієнтів у режимі реального часу та майже без допомоги операторів.

З початку 20 сторіччя почали активно розвиватися корпоративні інформаційні системи різного значення. Ще недавно звичним способом комунікації був стільниковий зв'язок. Багато компаній підтримували зв'язок з клієнтом за допомогою нього. Проте інтернет поступово витісняє інші способи комунікації, адже інформацію стало можливо передавати за допомогою бітового потоку даних у вигляді сигналів засобами електрозв'язку по каналу передачі даних. Цей сигнал далі оброблюється за допомогою засобів обчислювальної техніки.

Результати дослідження міністерства торгівлі США показали, що радіо знадобилося 30 років, щоб досягти аудиторії 50 мільйонів осіб, телебаченню – 13 років, а Інтернету – всього 4 роки. Обсяг даних, що пересилаються в Мережі, подвоюється кожні 100 днів [9].

Це дало людям можливість створювати вебресурси, у яких інформацію до людини можна доносити у вигляді зображень, тексту, відео чи анімацій. З'явилися візуальні та стилістичні засоби для привертання користувача до певної інформації, що можна назвати частиною поняття інтерфейсу.

Протягом останніх років відбулися значні зміни у тому, як ми створюємо програмне забезпечення. Цей еволюційний процес торкнувся не тільки серверної частини, а й інших частин з масовим переходом до використання доступних хмарних технологій та широким спектром технологій зберігання

даних, відмінних від традиційних систем управління БД. Багато в чому змінився і підхід до створення інтерфейсів користувача.

Великий акцент сьогодні змістився саме на вебпрограми. Вони продовжують, як і раніше, використовуватись для просування публічних інтернет-ресурсів. Але що важливіше, вебдодатки стають одним з основних способів реалізації рішень значно ширшого спектра завдань. Це і корпоративні послуги, які раніше більшою мірою реалізовувалися через настільні додатки (Desktop Applications). Розробка мобільних додатків також багато в чому прагне використати всю міць сучасних браузерів. Навіть ті програми, які виглядають і встановлюються як класичні настільні програми, насправді технічно є вебпрограмами, запущеними в браузері, позбавленому класичних елементів керування та навігації в Інтернеті. Прикладами таких програм зараз є Skype і Slack, Discord. Реалізація додатків у вигляді вебрішень дозволяє зробити їх кросплатформним, що дозволяє запустити їх на будь-якій із сучасних операційних систем, як і в основному на будь-якому сучасному браузері.

Інтернет дозволяє спеціалістам пропонувати свої послуги віддалено та майже незалежно від технічної частити чи засобу обробки інформації. Фрілансери стали новою категорією працівників, проте їхні можливості залежать від виду діяльності, ступеня професіоналізму, способу просування, країни проживання, законодавчої бази. Це призводить до нерівномірного розвитку віддаленої роботи в різних країнах.

1.2.2 Актуальність фрілансової біржі

Розвиток інтеграційних процесів у різних сферах діяльності вивів взаємодія між людьми на новий рівень. Зокрема, у суспільстві формуються нові економічні системи, наприклад, шерингова економіка чи економіка сумісного споживання (ЕСП). Вона заснована на співпраці та участі людей у житті один одного. Основний принцип – доступ до користування благом, а не володіння ним. Найважливішим двигуном розвитку економіки спільного споживання став

Інтернет, оскільки у ньому з'явилася величезна кількість шерінгсервісів – місць, де люди можуть зв'язатися та домовитись про співпрацю [9].

Постачальниками товарів та послуг можуть бути як окремі особи (продавці на сайтах оголошень, майстри, фрілансери), так і цілі компанії (каршеринги, послуги оренди). Найважливішим двигуном розвитку економіки спільного споживання став Інтернет, оскільки у ньому з'явилася величезна кількість шерінгсервісів – місць, де люди можуть зв'язатися та домовитись про співпрацю. Постачальниками товарів та послуг можуть бути як окремі особи (продавці на сайтах оголошень, майстри, фрілансери), так і цілі компанії (каршеринги, послуги оренди). Шерінгова економіка відкриває перед споживачами широкі можливості [10].

Очікується, що до 2025 року економіка спільного споживання сягне 335 мільярдів доларів. Компанії, що працюють в економіці спільного споживання, зростуть на 2133 % за 12 років [10].

До 2021 року понад 86 мільйонів американців використовуватимуть економіку спільного споживання, а краудфандинг зросте на 196,36 млрд. доларів у період з 2021 по 2025 рік [10]. За перші три місяці пандемії Covid-19 кількість бронювань на Airbnb впала на 53 %. 28 % користувачів готові поділитись своєю електронікою [10]. Спільне використання житла є найменш бажаною діяльністю в економіці спільного споживання. Головна проблема – довіра: щоб сервіси для обміну успішно працювали, користувачі повинні довіряти один одному. У цьому плані бізнес «від споживача до споживача» складніше, ніж від організації до споживача: компанія не несе особистої відповідальності за учасників, особливо коли їхні мільйони. Платформи у різний спосіб намагаються підвищити прозорість своєї діяльності: від різних рейтингів оцінки до посилення процедури підтвердження особи або автоматичного страхування [10].

Цей спосіб організації ринку таїть у собі не лише великі можливості, а й серйозні проблеми. Тим не менш, шерінг уже зараз став невід'ємною частиною багатьох сфер життєдіяльності, зокрема праці. Це дозволяє розвиватися

самозайнятому населенню. Самозайняті фахівці стали новим різновидом працівників.

Сучасні засоби виробництва та комунікації (комп'ютерна техніка, програмне забезпечення, Інтернет) дозволяють фрілансерам не тільки реалізовувати свій людський капітал самостійно (не вступаючи в традиційні трудові відносини), а й працювати практично на будь-якій відстані від місця, де необхідні результати їхньої діяльності. Останнє явище відоме як віддалена робота. Більше того, поява компаній, що спрощують взаємодію між користувачами, в деяких місцях стикається з громадським опором. Незважаючи на наявність як можливостей, так і проблем, шерінг уже зараз став невід'ємною частиною багатьох сфер життєдіяльності, зокрема праці. ЕСП дозволяє розвиватися самозайнятому населенню [10].

Як впливає з доповіді Міжнародної організації праці та Європейського фонду щодо покращення умов життя та праці, обсяг працівників, які працюють віддалено, продовжує зростати: у середньому в країнах Євросоюзу частка персоналу, що працює дистанційно, досягає 17% від обсягу всіх зайнятих, найбільш високі показники віддаленої роботи за межами Європи зафіксовані в США та Японії – 37% та 32% працівників відповідно [10].

1.3. Кількісні та якісні показники об'єкта

Цей розділ присвячений аналізу кількісних та якісних показників, які дадуть змогу орієнтуватися, для кого саме розробляється програмне забезпечення та на чому потрібно концентруватися.

Далі наводяться короткі відомості з ефективності віддаленої роботи у Америці, що підтверджує актуальність віддаленої роботи. Були опитані 10 000 співробітників Інститутом економіки Беккера Фрідмана в університеті Чикаго, які заявили, що вважають, що вони працюють вдома так само продуктивно, як і в офісі. Приблизно 30% респондентів сказали дослідникам, що вони більш продуктивні та завершені, працюючи вдома. Ця ж дослідницька група

підрахувала, що час у дорозі реєструється на 62,4 мільйона годин на день, а сукупна економія часу перевищує 9 мільйонів годин, виявлення з відхиленням від березня 2020 до вересня 2020 року [11]. Звіт Owl labs за 2021 показав, що 55% респондентів кажуть, що працюють віддалено більше годин, ніж у майбутньому в офісі [11]. Цей звіт показав, що тільки 36 % людей вважають, що офіс найкраще підходить для індивідуальної роботи [11]. 32 % опитаних Owl Labs заявили, що звільняться з роботи, якщо не зможуть взяти на себе роботу віддалено [11]. За даними сайту приблизно 1,1 мільярдів людей зайняті у сфері фрілансу [12]. На основі проаналізованих даних [13] побудовано рисунок 1.1, де показана демографічна ситуація у Америці за 2019 рік серед фрілансерів.

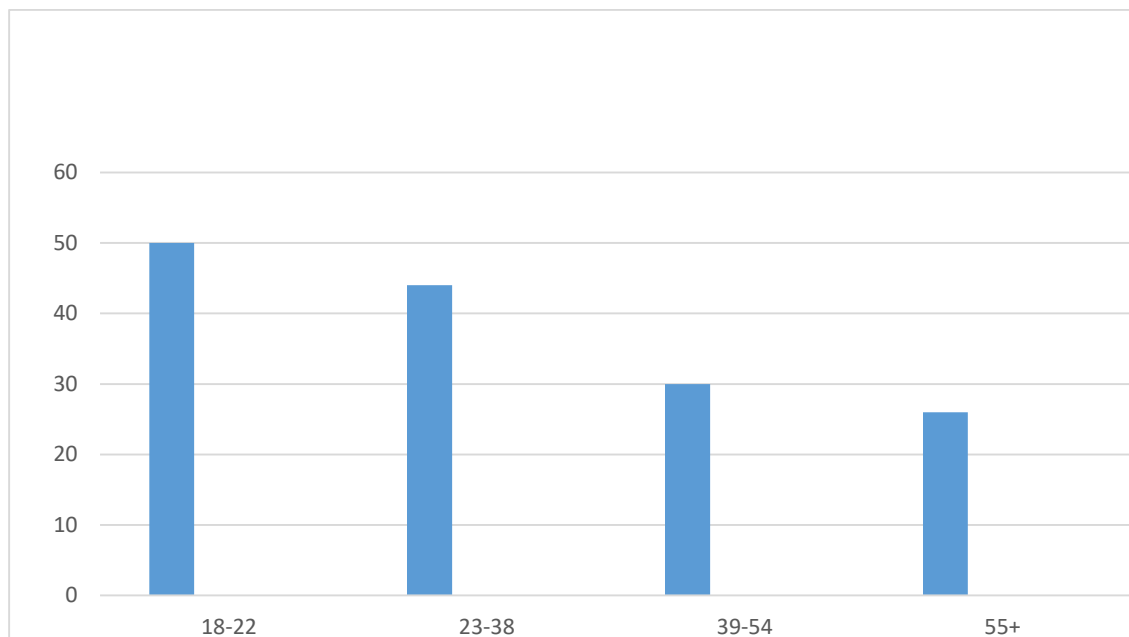


Рисунок 1.1 – Демографічна ситуація в Америці за 2019 рік

На основі даних [14], побудовано рисунок 1.2, де наводиться демографічна статистика для Великобританії.

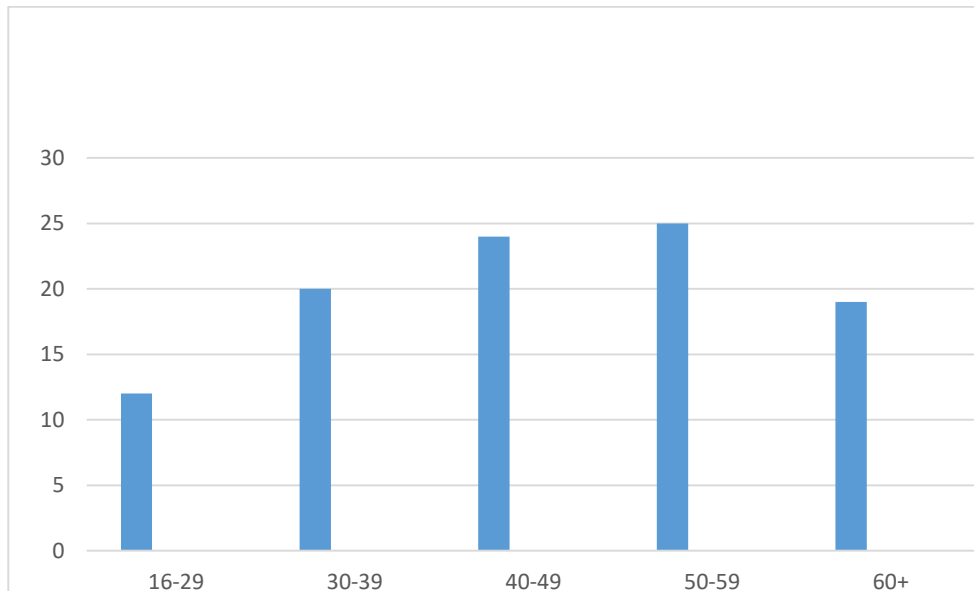


Рисунок 1.2 – Демографічна ситуація в Великобританії за 2019 рік

Далі на рисунку 1.3 приводяться дані Євростату [15]. На цьому рисунку розглядаються висококваліфіковані фрілансери, що дає змогу зрозуміти, на яких країнах потрібно зосереджувати свою увагу.

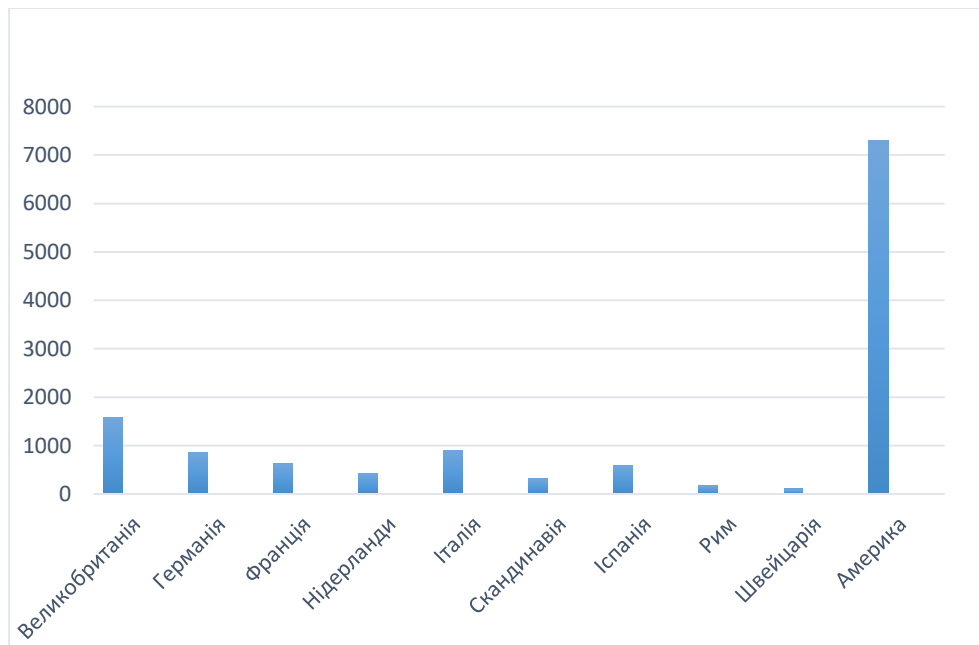


Рисунок 1.3 – Кількість висококваліфікованих фрілансерів

Із графіку можна побачити, що загальна кількість висококваліфікованих фрілансерів 2,8 мільйона. Населення яке працює у цифровій сфері становить

приблизно 13 мільйонів осіб (за даними Євростату), що становить близько четвертої частини талантів у Європі [15].

Виходячи з графіку, можна побачити, що найбільшу кількість має Америка, потім Великобританія. Це обумовлено тим, що саме це поняття та спосіб заробітку з'явився в Америці. Також в деяких країнах розповсюдження фрілансу, як способу самоорганізації, він є наслідком високого безробіття, прикладом виступає Італія, Іспанія. На наступному рисунку 1.4 зображена доля ринку у фрілансі за 2020 рік [15].

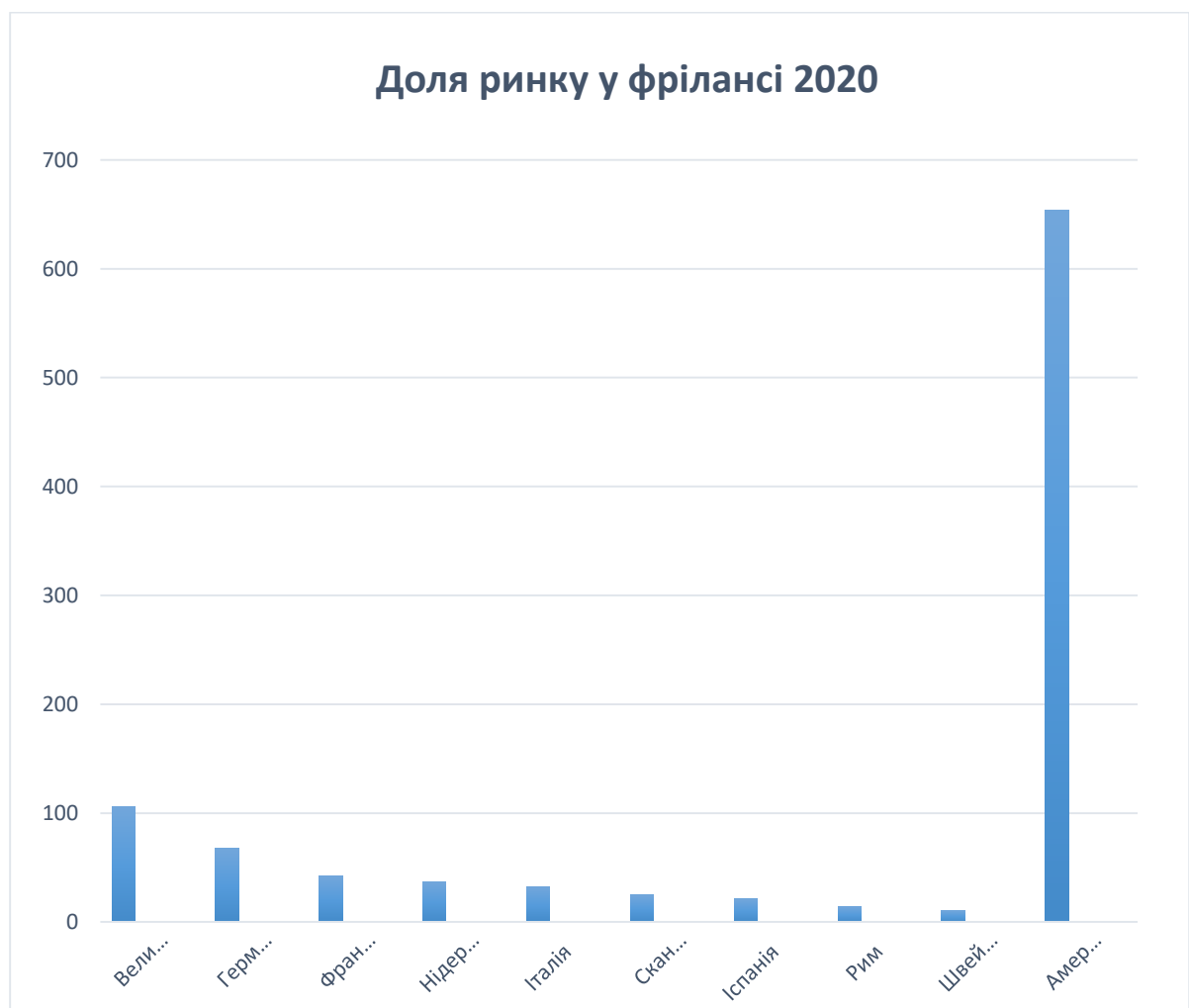


Рисунок 1.4 – Доля ринку у фрілансі 2020

Виходячи з наступного порівняння, у Європі доля ринку становить 355 мільярдів євро. У Америці – 654 мільярда доларів.

На зростання ринків фрілансу впливають три фактори:

- економічне середовище;
- зрілість позаштатної економіки;
- правове середовище [15].

1.3.1 Економічне середовище

Першим фактором є те, що більшість людей будуть хотіти стати фрілансером, якщо умови для роботи будуть більш сприятливими. Зростання ВВП є сильною рушійною силою для зростання у населенні фрілансерів. Ще одним фактором виступає безробіття, яке безпосередньо впливає на середні ціни фрілансерів. У той час як середня зарплата штатного працівника не піддається коливанням, ставка фрілансера може значно коливатися в залежності від попиту та пропозиції [15].

Наприклад в Іспанії зарплати нижче ніж у Франції у середньому на 25 % за аналогічну роботу, проте щоденні ставки фрілансерів на 50 % нижчі, ніж у Франції [15]. У Німеччині щоденні ставки фрілансерів на 30 % вищі, ніж у Франції, тоді як штатні працівники мають доходи на 25 % більші [15]. На наступному рисунку 1.5 зображено порівняння оплати фрілансерів у обговорених країнах [15].

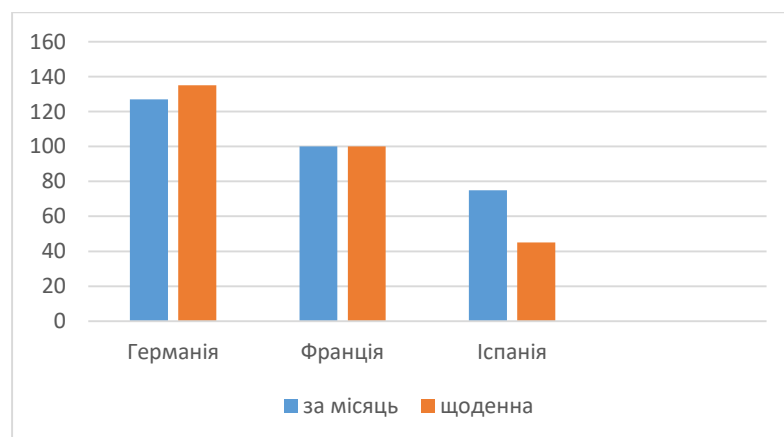


Рисунок 1.5 – Порівняння оплати фрілансерів

Криза COVID стала чудовим прикладом того, як поведінка може змінитися. Хоча впевненість фрілансерів у майбутньому залишається високою, кількість фрілансерів, які шукають роботу на повний робочий день, зросла після того, як вибухнула криза [15].

Ставлення клієнтів до фрілансу важче виміряти, і його можна посправжньому оцінити лише шляхом співбесіди з керівниками бізнесу, такими як менеджери ІТ-проектів, директори із закупівель тощо [15].

Зрілість клієнта є ще одним потужним чинником зростання. Оскільки зараз все більше компаній бажають працювати з фрілансерами, все більше і більше традиційних консультантів також обирають шлях, у якому фрілансери виступають прихованими робітниками агентства або консалтингової фірми. Вони часто працюють над тими ж проектами, що й фрілансери, в тих самих командах і за однакову щоденну ставку. Однак фрілансери не мають структурних витрат консалтингової фірми. Якщо клієнти починають працювати безпосередньо з фрілансерами, вони допомагають консультантам-співробітникам зробити «стрибок віри» та перейти до незалежності. Потім, коли все більше і більше старших експертів стають фрілансерами на ринку, все більше і більше компаній починають приймати нову реальність, таким чином підтримуючи цю петлю зворотного зв'язку

1.3.2 Зрілість ставлення працівників до фрілансу

Бути фрілансером означає бути підприємцем. Не всі створені для підприємництва чи ринку, який вже наповнений фрілансерами. Так, наприклад, у Великобританії та Нідерландах, серед активного населення 6,3% і 7,2% фрілансерів відповідно [15]. У Франції та Іспанії для порівняння фрілансери становлять 3,3 % і 3,6 % населення. Це половина частки більш зрілих фрілансових економік [15].

Ця початкова точка сильно вплине на потенційне зростання ринку. Такі країни, як Франція, Іспанія та Німеччина, мають більше простору для зростання ринку фрілансу [15].

Окрім дуже різної ринкової динаміки, також помітно, що типовий фрілансер дуже відрізняється від країни до країни. Хоча вибір завжди є сильною рушійною силою для переконання людей перейти з повної зайнятості на фріланс, що є досить важкою задачею, розбіжності залишаються в тому, що хочуть обрати фрілансери і ким вони хочуть бути [15].

1.3.3 Правове середовище

Франція є першою у рейтингу правого захисту фрілансерів. Фрілансери мають багато різних варіантів виставлення рахунків своїм клієнтам, Франція ввела значно спрощений статус у 2011 році: мікропідприємство (автопідприємство) дозволяє фрілансеру отримати реєстраційний номер менше ніж за 48 годин і виставляти клієнтам рахунок з обмеженим оподаткуванням до 70 000 євро на рік. Безперечно, що цей статус сприяв зростанню економіки талантів у Франції [15].

Фрілансери у Великобританії та Іспанії також виграють від відносно привабливих правових рамок. У Великобританії можна виставляти рахунки клієнтам під особистим ім'ям, без реєстрації, тоді як в Іспанії фрілансери зазвичай обирають автономний статус. Незважаючи на те, що отримати цей статус дуже просто, він має один серйозний недолік: мінімальна сума платежів за соціальне страхування становить 287 євро на місяць, що робить початок кар'єри фрілансера потенційно ризикованим, оскільки доведеться платити цю суму, навіть якщо рахунки низькі [15].

1.4 Огляд існуючих аналогів

1.4.1 Фрілансова біржа «Upwork»

Платформа Upwork — це онлайнринок для фрілансерів, які працюють у таких сферах, як письмо, графічний дизайн та веброзробка. Сайт допомагає професіоналам знаходити проекти, спілкуватися з клієнтами та отримувати за це гроші [16].

1.4.1.1 Опис

В цілому, платформа Upwork спрощує пошук роботи [16]. Проте інтерфейс цієї платформи занадто перевантажений функціоналом, хоча на платформі існують відеозаписи, які дещо допомагають вирішувати цю проблему. В цілому на платформі добре продумані функції, що допомагають фрілансерам.

1.4.1.2 Переваги

Переваги даної платформи наступні:

- надає весь необхідний інструментарій для створення особистого профілю з портфоліо;
- надає доступ до величезної кількості доступних робіт та замовників;
- надає зручний фінансовий інструмент та автоматизує отримання грошей із клієнтів. А також дає можливість вибирати між двома типами оплати
 - оплата за годину та фіксована оплата за замовлення;
 - у разі погодинної оплати надає програму для комп'ютера, яка автоматично відстежуватиме час і автоматично списуватиме з клієнта гроші;
 - у всіх типах замовлень виступає гарантом отримання грошей за успішне виконання замовлення, тобто клієнт завжди сплатить за роботу;

- надає всю історію замовлень та рейтингову систему як для клієнтів, так і для фрілансерів. Таким чином можна уникнути роботи з неадекватними клієнтами з поганими відгуками;
- є система різних градацій як фрілансера, при дотриманні певних умов дає доступ до різних нових плюсів;
- дає доступ до великої кількості статистики за заробітками, напрацьованими годинами і т.д.;
- є вбудований месенджер для спілкування з клієнтами, можна використовувати скріншоти з нього для вирішення спірних ситуацій [16]-[17].

1.4.1.3 Недоліки

Недоліки цієї платформи наступні:

- за свої послуги Upwork бере величезні комісії, ситуація з тим, скільки потрібно платити, постійно погіршується. Градація того, скільки у відсотках клієнт заплатить платформі, залежить від того, скільки всього грошей він заробив з конкретним клієнтом;
- часто буває складно перевірити, чи правдивий текст на платформі фрілансера щодо досвіду, освіти та навіть завершених проєктів (за межами платформи);
- тривалий процес вибору відповідного фрілансера, спілкування та аналізу виконання роботи над проєктом;
- величезна конкуренція змушує нових фрілансерів братися за не зовсім актуальні проєкти, платформа Upwork має погану репутацію щодо захисту фрілансерів. Ця платформа була помічена в абсолютно безпричинному блокуванні фрілансерів з формулюванням «до з'ясування обставин»;
- дуже складно працювати на цій платформі спочатку. Складна ситуація з початковим відшкодування за роботу на цій платформі [16]-[17].

На рисунку 1.6 зображена головна сторінка платформи «Upwork».

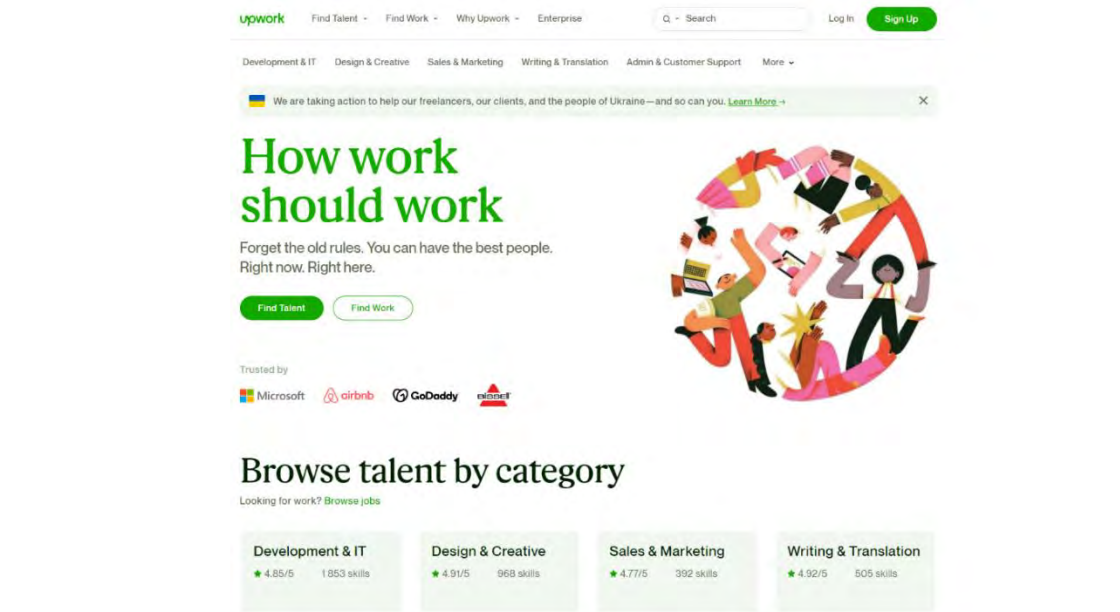


Рисунок 1.6 – Головна сторінка «Upwork» [18]

1.4.2 Фрілансова біржа «Fiverr»

Платформа Fiverr – це найбільший у світі ринок цифрових послуг, де приватні особи чи компанії можуть знайти фрілансерів з навичками, які відповідають їхнім потребам, і вважається сайтом «мікрозавдання», де можна виконувати відносно швидкі невеликі одноразові завдання [19].

1.4.2.1 Опис

Платформа Fiverr: назва початково походить від вартості 5 доларів за завдання, а тепер вона варіюється до 995 доларів за завдання. Існує широкий спектр послуг, які можна продавати на Fiverr, починаючи від професійно кваліфікованих послуг і закінчуючи дивними завданнями [19].

1.4.2.2 Переваги

Переваги даної платформи наступні:

- Fiverr представляє зручне універсальне рішення для керування та пошуку позаштатної роботи;
- має велику свободу в ціноутворенні замовлень;
- Fiverr надає широкі можливості для вивчення нових навичок та покращення сервісу;
- реєстрація для фрілансерів безкоштовна.

1.4.2.3 Недоліки

Недоліки цієї платформи наступні:

- Fiverr – висококонкурентний майданчик для фрілансерів;
- Fiverr стягує комісію з продажу;
- початківцям може бути складно бути поміченим, тому що це дуже конкурентне середовище – алгоритм Fiverr просуває фрілансерів, які вже мають хороші відгуки [20].

На рисунку 1.7 зображена головна сторінка платформи «Fiverr».

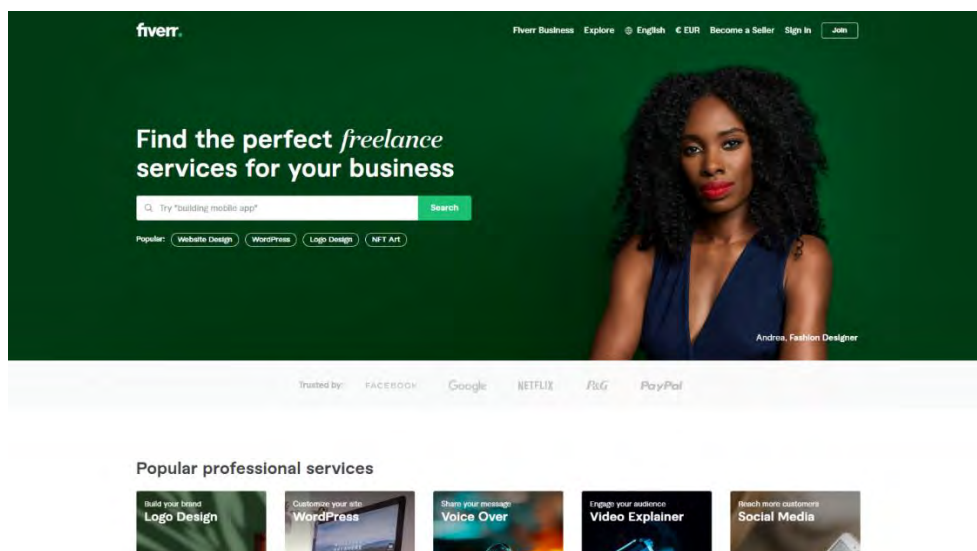


Рисунок 1.7 – Головна сторінка «Fiverr» [21]

1.4.3 Фрілансова біржа «99designs»

Платформа «99designs» – компанія з цифрового графічного дизайну. Це допомагає компаніям знайти потрібний високоякісний дизайн, зв'язуючи їх із позаштатними графічними дизайнерами. 99designs займається лише дизайном, але не пропонує ніяких недизайнерських послуг, таких як друк. [22].

1.4.3.2 Опис

Платформа «99designs» – це застосунок для створення електронних графічних видів роботи. Проте функціональність яку має ця платформа має досить багато вбудованих інструментів для редагування зображень, що робить її унікальною серед інших.

1.4.3.3 Переваги

Переваги даної платформи наступні:

- великі ціни у призових фондах. Наприклад, вартість логотипу становить зазвичай 300-400 доларів [23];
- щодня оновлюється стрічка конкурсів, тож завжди є великий вибір проєктів [23].

1.4.3.4 Недоліки

Недоліки цієї платформи наступні:

- крадіжка конкурсних робіт. Якщо роботи учасників можуть бачити всі, то навіть якщо вони представлені у форматі jpg, нічого не заважає зробити потім свій дизайн на основі знайденої ідеї;
- виведення грошей обмежено за варіантами: Moneybookers, PayPal, AlertPay або Western Union [24];

– класифікація фрілансерів за рівнями: початковий рівень, сертифікований та платиновий. Розподіл між ними незрозумілий. Так чи інакше, не можна брати участь у всіх конкурсах, опублікованих на сайті, а значить не можна підвищувати кваліфікацію [24].

На рисунку 1.8 зображена головна сторінка платформи «99designs».

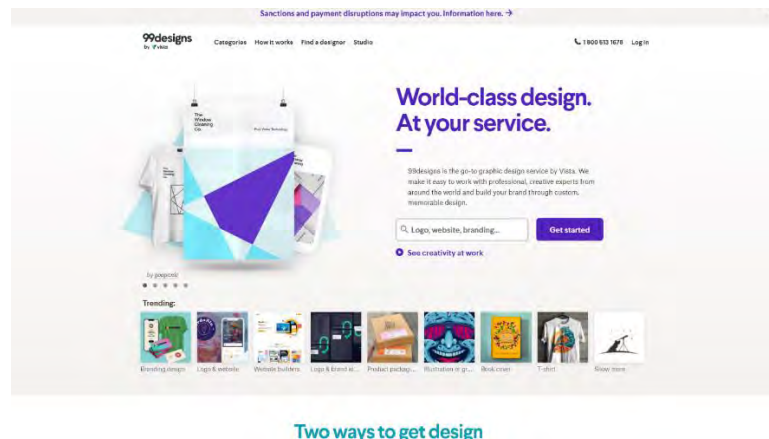


Рисунок 1.8 – Головна сторінка «99designs» [25]

1.4.4 Фрілансова біржа «freelancer»

1.4.4.1 Визначення

Платформа «Freelancer» – найбільший у світі ринок фрілансу та краудсорсингу за кількістю користувачів і проєктів, який об’єднує понад 59 689 096 роботодавців і фрілансерів по всьому світу з більш ніж 247 країн, регіонів і територій [26].

1.4.4.2 Опис

Сайт також дозволяє учасникам проводити та брати участь у конкурсах, за які як винагорода пропонуються грошові призи, а також фрілансери та роботодавці створюють профілі на сайті у міру того, як вони пропонують, виграють та виконують роботу, а також пишуть та отримують відгуки про людей, з якими вони працюють або для яких вони працюють [27]. Учасники

сайту одержують кінцеву кількість ставок для використання на сайті, які періодично поповнюються, пропонується низка варіантів облікового запису, від безкоштовних облікових записів до професійних підписок [27].

1.4.4.3 Переваги

Переваги даної платформи наступні:

- конкурси із гарантованою виплатою;
- рівні, кредити та сумнівні бонуси;
- велика цінність іспитів, які є дешевими;
- за виведення біржа не бере гроші [27].

1.4.4.4. Недоліки

Недоліки цієї платформи наступні:

- замало проєктів;
- своєрідний спосіб стягнення комісії, який знімають відразу після підписання контракту перед виконанням роботи;
- незахищеність фрілансерів та користувачів. [27].

На рисунку 1.9 зображена головна сторінка платформи «freelancer».

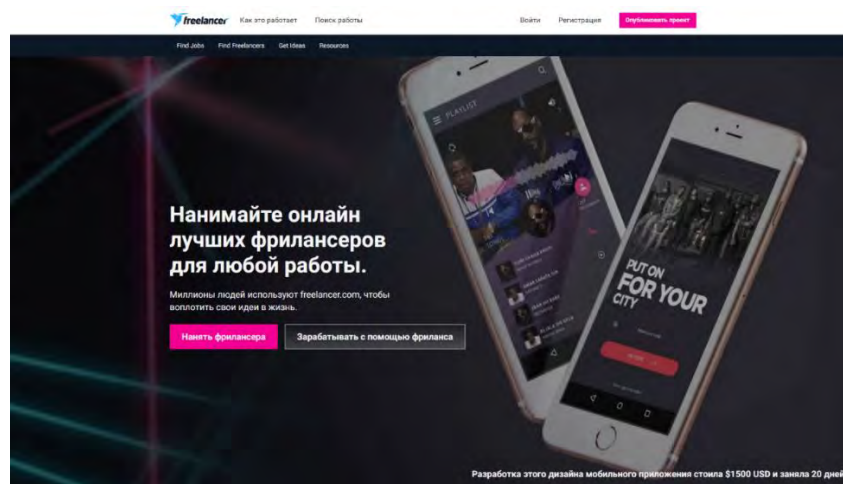


Рисунок 1.9 – Головна сторінка «freelancer» [28]

1.4.6 Фрілансова біржа «Guru»

1.4.6.1 Визначення

Платформа Guru.com є об'єктом онлайнового простору для фрілансерів для окремих фрілансерів так і для великих команд [32]. Інтерфейс має багато функціональності, що значно спрощує життя користувачам.

1.4.6.2 Опис

Платформа Guru.com – компанія розташована одразу по всьому світу і користувачі мають змогу звертатися до іншомовних колег без зайвих проблем. Цей сервіс вже пройшов шлях розміром у 10 років, що дало змогу створити багатофункціональний сервіс [32].

1.4.6.3 Переваги

Переваги даної платформи наступні:

- дуже досвідчені фрілансери;
- інтуїтивно зрозумілий інтерфейс;
- програма для відстеження часу, проведеного за роботою;
- великі знижки для користувачів, що оформили передплату на сервіс.

1.4.6.4. Недоліки

Недоліки цієї платформи наступні:

- великі комісії за транзакції та дорога щомісячна передплата;
- складна система розрахунку з користувачами;
- немає живого чату [33].

На рисунку 1.11 зображена головна сторінка платформи «Guru».

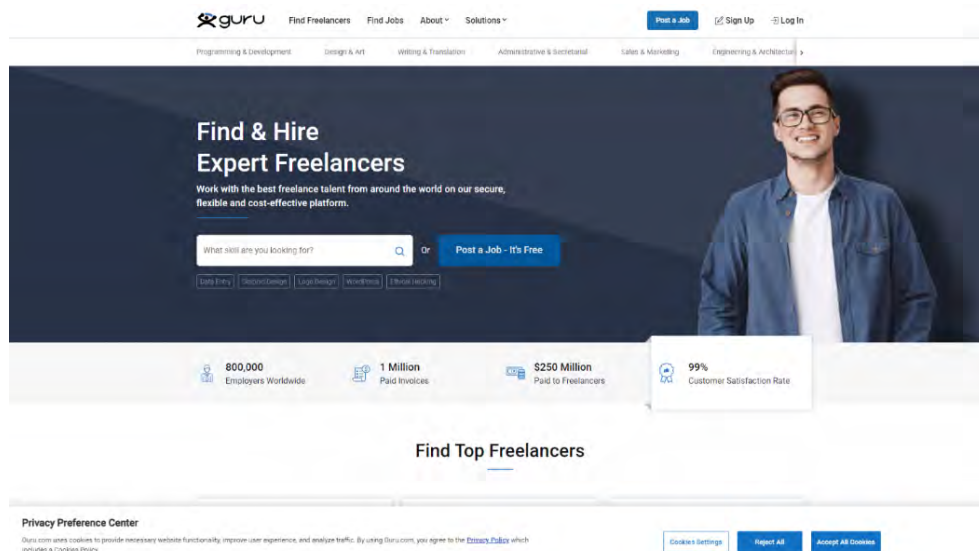


Рисунок 1.11 – Головна сторінка «Guru» [34]

1.4.7 Фрілансова біржа «ServiceScape»

1.4.7.1 Визначення

Платформа «ServiceScape» – це платформа для позаштатного ринку та аутсорсингової платформи, яку підприємства використовують для пошуку позаштатних редакторів, перекладачів, графічних дизайнерів та письменників [36].

1.4.7.2 Опис

Протягом двох десятиліть «ServiceScape» об'єднує компанії з професійними фрілансерами та створив мережу досвідчених позаштатних редакторів, письменників, перекладачів та графічних дизайнерів. Платформа проста та зручна у використанні, спілкування відкрите та зрозуміле, а профілі детальні.

Відгуки на вебсайті та відгуки клієнтів в інших місцях свідчать про те, що компанія надає чудовий сервіс. Якщо віддається перевага проводити конкурси та отримувати багато варіантів і пропозицій щодо проєкту, «ServiceScape» не підходить для цієї роботи [35].

1.4.7.3 Переваги

Переваги даної платформи наступні:

- зв'язок фрілансерів з клієнтами;
- забезпечити оплату фрілансерам за виконану роботу;
- пропонування методів зворотного зв'язку як для фрілансерів, так і для клієнтів [35].

1.4.7.4. Недоліки

Недоліки цієї платформи наступні:

- відносно високі комісії, що знімаються з заробітку;
- необхідність платити за підвищення ставки в списку фрілансерів робить це ще дорожчим [35].

На рисунку 1.12 зображена головна сторінка платформи «ServiceScape».

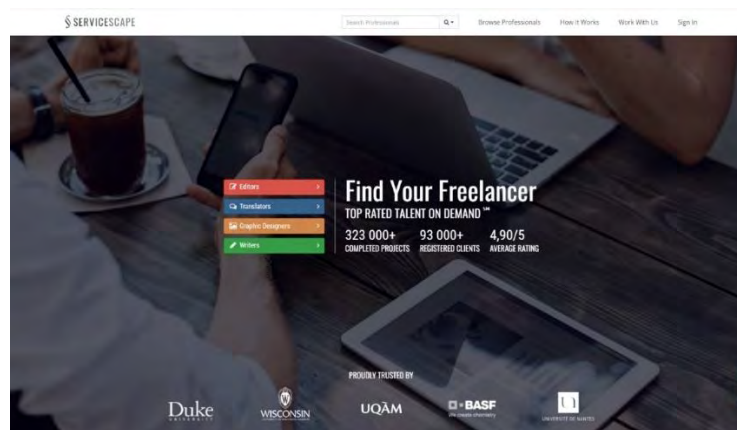


Рисунок 1.12 – Головна сторінка «ServiceScape» [36]

1.5 Порівняльна характеристика існуючих аналогів та розроблюваного програмного продукту

Виходячи з пункту 1.4, можна зробити порівняльну характеристику розроблюваного застосунку та аналогів.

Основним завданням фрілансевої біржі є налаштування взаємозв'язку замовника та самозайнятого працівника (фрілансера). Тому важливим критерієм оцінки є взаємодія користувачів за допомогою чату. Під цим поняттям повинно буде аналізуватися взаємодія користувачів за допомогою рисунків (смайлів), можливість прикріплювати та зберігати файли. Наступними важливими чинниками будуть виступати привертання уваги замовника за допомогою великої кількості показників та зручний набір фільтрів для пошуку фрілансера. У самозайнятого працівника повинна бути можливість заповнити всю необхідну інформацію про себе, щоб зацікавити потенційного замовника. У свою чергу у замовника повинна бути найглибша система фільтрів, за допомогою якої замовник зможе знайти для себе найбільш підходящу людину для виконання роботи.

Потрібно також виокремити критерій інтерфейсу, який теж є доволі складною задачею, яка потребує багато зусиль. Потрібно розглянути палітру кольорів та дотримуватися правил візуального потоку, візуальної ієрархії, групувань та вирівнювань елементів. Інтерфейс не повинен бути перевантаженим. Повинні бути створені шаблони, мокапи які будуть використовуватися у програмі, щоб інтерфейс був максимально одноманітним та інтуїтивно-зрозумілим.

Також позитивною рисою розроблюваного застосунку буде те, що користувач може виступати як фрілансером, так і замовником. Передбачена можливість швидкої реєстрації та зміни ролі.

У таблиці 1.1 буде виокремлені ключові критерії по яким будуть порівнюватися застосунки.

Таблиця 1.1 – Порівняльна характеристика аналогів і програмного застосування

Критерій оцінки	Аналоги				
	Upwork	Fiverr	99designs	Service Scape	Дипломна розробка
1	2	3	4	5	6
Чат	+/-	+/-	-	+/-	+/-
Прикріплювання файлів	+	-	-	+	+
Інтерфейс	+/-	+/-	+	+	-
Конкурси	+	+	-	-	-
Профіль фрілансера	+	-	-	-	+
Регулювання відносин	+	-	-	+	-
Швидка зміна ролі	-	-	-	-	+
Алгоритм	-	+	-	-	-
Пошук фрілансерів	+	-	-	-	+

1.6 Висновки за першим розділом

Отже виходячи з виконаного аналізу проблеми, можна зробити висновок, що фріланс дуже активно розвивається, чому сприяла пандемія коронавірусу на поточному етапі. Також є тенденція щодо віддаленої роботи загалом, а кількість замовлень у сфері фрілансу збільшується.

Поняття фрілансу досить нове, а правовий захист дуже слабкий. У деяких державах це ніяк не регулюється, а податкова система відсутня, що здійснює негативний вплив на економіку держави.

Також великою проблемою для початківців у цій сфері виступає репутація, яку дуже складно заробити, як і досвід. Якщо порівнювати робочий день у компаніях та фрілансових біржах, то на такій людині лежить набагато більше зобов'язань.

Виходячи з кількісних та якісних показників, у фрілансі майже задіяно усі вікові категорії людей. Найбільшу популярність фріланс має у Америці та Великобританії, також це можна сказати і про кваліфікованих фрілансерів. Існує чимало аналогів та бірж фрілансингу, які безумовно мають свої переваги та недоліки.

За даними, які були наведені у цьому розділі, можна зробити висновки, що найбільше самозайнятих працівників зустрічається саме у Америці та Великобританії. Тому під час розробки потрібно звертати увагу на сайти цих країн. Також потрібно зважувати, що велика кількість фрілансерів має значний вік, тому потрібно думати про зрозумілий інтерфейс з допомогою для розуміння.

Потрібно звернути увагу на проблему правового захисту та можливість взаємодії з державою для сплати податків.

Завдання дипломної кваліфікаційної роботи було наступне:

- аналіз предметної області;
- визначення проблемних зон предметної області;
- визначення вимог до програмного забезпечення;

- проєктування програмного забезпечення біржі фрілансу;
- реалізація і налагодження серверної частини ПЗ.

2 ПРОЄКТУВАННЯ ПРОГРАМИ

2.1 Вибір мови програмування

Фактично вибір мови програмування для написання вебзастосунку залежить від цілей розробника. На сьогоднішній день список мов програмування дуже великий: Python, Java, PHP, Go, C#, Ruby, Node.JS, Golang (Go), Perl, JavaScript. Також потрібно звертати увагу на кількість фреймворів та бібліотек та кількість доступності інформації. Також важливим фактором є кількість документації та час затрачений на вивчення синтаксису та особливостей мови. Лідером серед мов програмування для веброзробки виступає JavaScript та надбудова над цією мовою TypeScript. Ці мови перші по популярності для розробки вебзастосунків. Варто зазначити, що критерій швидкості компіляції, використання ресурсів та оперативної пам'яті взятий без суттєвих досліджень, що не робить його основним орієнтиром для вибору мови програмування: дуже багато залежить від самого програміста та його знань, що буде суттєво впливати на продуктивність застосунку.

Майже усі мови мають JIT компілятор, що суттєво зменшує час на компіляцію. Найповільнішою мовою є python, попри свою багату функціональність та велику кількість бібліотек та фреймворків. Незважаючи на використання віртуальної машини, Java все ж таки швидше за свої аналоги, такі як C#. При компіляції Java використовує байткод, який чимось схожий на IL код мови C#. Мова Go йде на рівні з C# по продуктивності. Майже всі мови мають повну кросплатформеність.

За критерієм складності мови для вивчення, що є достатньо важливим, явним лідером виступає python. Він має інтуїтивно зрозумілий набір операторів та синтаксичних конструкцій, який є достатнім для вирішення багатьох завдань. Також ця мова є динамічною, що інкапсулює типи, що значно спрощує її вивчення.

Окремо варто розглянути мову JavaScript: сучасна версія цієї мови перейняла дуже багато особливостей від попередніх версій, що ускладнює її

вивчення. Також відсутність строгої типізації дає можливість розробникам створювати код без обмежень, що потім може ускладнити подальшу розробку коду. Хоча ця мова і має багато особливостей, це можливо компенсувати широкою популярністю і величезною кількістю інформації. Мова JavaScript та обгортка TypeScript має величезну кількість фреймворків та бібліотек, як і попередні мови програмування. Якщо вебзастосунок передбачає якісний фронтенд, то без цієї мови неможливо обійтися. На цій мові програмування можна писати, як фронтенд так і бекенд частину застосунку. Дуже популярною платформою для написання такого застосунку виступає Node.js.

Якщо говорити про GoLand, то це відносно нова мова програмування, яка дуже сильно підтримуються Google. Вона є доволі простою та відносно середньою за продуктивністю. Проте у цій мові дуже замало функціональності, що робить досить складним вирішення задач, а кількість інформації є теж доволі невеликою.

Кожна з цих мов має сайти з документацією. Загалом найширша та найкраща документація у C#. Вона має величезну кількість прикладів та рішень для багатьох платформ, що в цілому може навіть замінювати сайт “stackoverflow”.

Найкращими мовами програмування за сумою характеристик виступають C#, Java, JavaScript (обгортка TypeScript), тому потрібно зосередитися на них. Усі ці мови мають величезну кількість інформації у інтернеті. Також усі ці мови підтримують кросплатформеність.

Мова C# багата на технології (фреймворки та бібліотеки). У цій мові є платформи WebApi та MVC, які були створені під вебзастосунки. Також ця мова програмування має сильні інструменти, що спрощують роботу з БД, такі як Linq та EntityFramework. Також чималою перевагою є спрощення роботи з пам'яттю за допомогою Garbage Collector. Мова є дуже читабельною. Також ця мова дозволяє напряду працювати з ІЛ кодом, що дозволяє значно підвищувати продуктивність.

Мова Java має теж чимало технологій, однією з яких виступає Spring Boot. Ця мова також має збирача сміття, а її синтаксис схожий на C#. Насправді Java і C# – це прямі конкуренти, які запозичували особливості один у одного. Проте кількість підтримки та інформації для цієї мови менша. Мова компілюється у байткод, що теж дає змогу оптимізувати написаний код. Також мова запозичила більше особливостей та проблем з мови C++. Спрощена порівняльна характеристика наведена у таблиці 2.1.

Таблиця 2.1 – Аналіз мов програмування для створення “Freelance Exchange Software”

Мова програмування	Критерії оцінки мови програмування					
	Типізація	Швидкість	Кросплатформеність	Вивчення	Кількість інформації	Фреймворки та бібліотеки
1	2	3	4	5	6	7
JavaScript TypeScript	динамічна /слабка динамічна	швидше ніж середня	+	складня	Висока	Велика
Python	динамічна	повільно	+	середня	Висока	Велика
Java	статична	швидко	+	дуже складно	Висока	Середня

Продовження таблиці 2.1

1	2	3	4	5	6	7
C#	нестрога статична	швидше ніж середня	+/-	складно	Висока	Велика
PHP	слабка динаміч на	повільно	+	середня	Середня	Середня
Go	строга статична	середня	+	легко	Середня	Невелика

Отже враховуючи усі фактори, а також дуже значущий фактор досвіду, для цього випадку найкращим вибором є мова C# в якості основної мови програмування застосунку. Для фронтенд частини буде використовується JavaScript.

2.2 Проектування архітектури програмного забезпечення та огляд варіантів

Існує чимало архітектурних рішень. З них потрібно розглянути наступні:

- монолітна архітектура;
- багатошарова;
- оніон;
- мікросервісна;
- хелікс.

2.2.1 Монолітна архітектура

В монолітному проєкті уся програма зосереджена в одному проєкті, що у майбутньому буде ускладнювати розробку, в такому підході є проблема чіткої

роздільності та модульності програмного застосунку, окрім того неможливо розбити цей застосунок по контейнерах у докері, що може привести до проблем у інших розробників з розгортанням застосунку. Зі збільшенням застосунку буде збільшуватися й час розгортання, що може призвести до потенціальних помилок. Найголовнішою проблемою є те, що модулі такого застосунку буде неможливо перевикористати. Така архітектура буде підходити для дуже малих немасштабованих проєктів для власного використання. Також ця архітектура не потребує зайвих зусиль для чіткого розмежування коду, бібліотек, що значно може заощадити час. На рисунку 2.1 зображена схематична структура монолітної архітектури.



Рисунок 2.1 – Структура монолітної архітектури

2.2.2 Багатошарова архітектура

Багатошарова архітектура найбільш популярна серед інших. Класична модель складається з трьох шарів:

- рівень доступу до даних (сама БД);

- рівень бізнес-логіки;
- рівень представлення.

Кількість шарів у такій архітектурі визначає сам розробник, що робить її досить гнучкою, про що свідчать багато архітектур, які включають ідею багат шаровості, ця архітектура підтримує принцип чіткої розмежованості (SoC). На відміну від монолітної архітектури ця архітектура має чітке розмежування та має можливість перевикористання чи заміни модулів. Також з'являється можливість для розробки шарів на інших мовах програмування та розділення їх по різних серверах чи контейнерах докера, що дає у майбутньому можливість для масштабування такого застосунку, що у свою чергу відкриває можливість для балансування навантаження на сервері за допомогою Kubernetes. Також це робить шари такої архітектури ізольованими один від одного. Також у різних доповненнях виключається проблема циклічних посилань. Посилання на проекти, бібліотеки ідуть строго від рівня даних до рівня представлення. На рисунку 2.2 зображена схематична структура багат шарової архітектури.

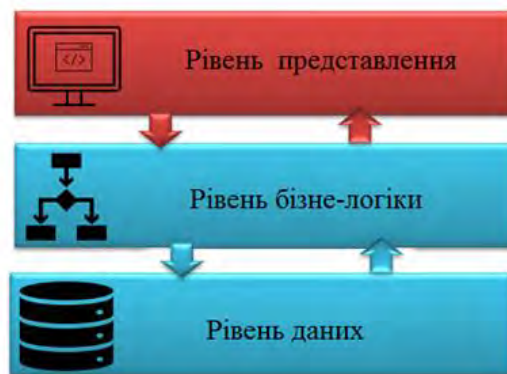


Рисунок 2.2 – Структура багат шарової архітектури

2.2.3 Onion-архітектура

Також варто розглянути Onion-архітектуру, яка в собі має структуру нашарування та в деякому роді є продовженням багат шарової архітектури.

Ідея цієї архітектури полягає у тому, щоб розмістити домен і служби для обробки даних в центрі програми. Цей рівень нічого не буде знати про інші нашарування та буде мати доменні моделі, які будуть використовуватися у наступних шарах. Кожен наступний рівень буде нашаровуватися на попередній і не може існувати окремо. Зазвичай така архітектура має наступні шари:

- доменний рівень;
- рівень репозиторієв або рівень взаємодії з даними;
- рівень сервісів або бізнес-логіки;
- рівень представлення.

На відміну від від багатошарової архітектури така архітектура не потребує власних моделей на кожному рівні, що значно зменшує кількість зусиль для написання коду, проте в свою чергу рівні не будуть ізольовані один від одного і це ускладнює масштабування. Це є середнім варіантом між монолітною та багатошаровою архітектурою. На рисунку 2.3 зображена схематична структура Onion-архітектури.

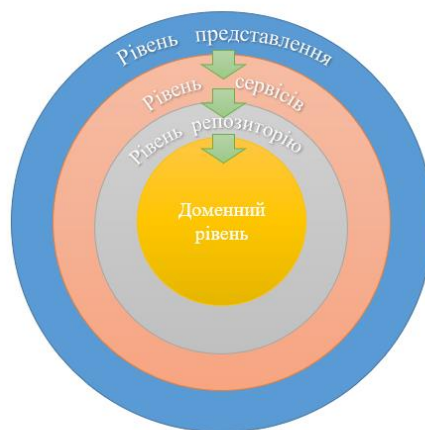


Рисунок 2.3 – Структура Onion-архітектури

2.2.4 Мікросервісна архітектура

Наступною архітектурою виступає мікросервісна. Це архітектура, яка зазвичай має у собі набір бізнес-функцій, які у свою чергу можуть бути розгорнуті як незалежні компоненти і можуть виступати окремими застосунками. Також ці мікросервіси можуть бути написані на різних мовах та використовувати різні підходи та технології. Кожен з таких мікросервісів працює у своєму процесі. Зазвичай їх поєднує лише програмний інтерфейс застосунку: у випадку вебзастосунку – це Rest. Цю архітектуру дуже легко розгортати, використовувати Docker чи Kubernetes для оркестрування контейнерами. Це найбільш сучасна та найдорожча за часом та ресурсами архітектура, що робить її використання доцільним лише для великих застосунків. На рисунку 2.4 зображена схематична структура мікросервісної архітектури.



Рисунок 2.4 – Структура мікросервісної архітектури

2.2.5 Helix-архітектура

Наступною архітектурою є Helix. Це доволі малорозповсюджена, проте доволі сучасна архітектура, яка є проміжним рішенням між ідеями мікросервісної архітектури та багат шарової. В ідеях цієї архітектури зроблений акцент на макроархітектуру. У цій архітектурі є три основні рівня:

- рівень залежностей технологій інструментів (Foundation);
- рівень функціональностей або бізнес логіки (Feature);
- рівень представлення (Project).

Кожен з цих рівнів може мати ще підрівні. Так декілька цих рівнів з кожного шару можуть представляти функціональну групу, яка потрібна для виконання якоїсь частини функціональності. На рівні залежностей мають розміщуватися потрібні технології, розширення, функції і т.д., що потрібні будуть для використання на рівні функціональності. Це дає змогу для перевикористання цих модулів у майбутньому та робить таку архітектуру максимально розподіленою і чіткою. На рівні функціональності проходять основні операції з даними, цей рівень може бути і мікросервісом. Останнім рівнем виступає рівень представлення, який буде у собі містити стартові файли для запуску деяких рівнів з шару функціональності. Також цей рівень може мати представлення для відображення результату чи результатів з попереднього шару. Така архітектура підходить для масштабування, докер контейнерів, оркестрування контейнерами. Також варто зазначити, що у такій архітектурі залежності йдуть строго від рівня залежностей до представлення. А кожен рівень у шарі не повинен знати про інший рівень, що робить таку архітектуру надзвичайно стійкою до збоїв та легкою до розгортання.

На рисунку 2.5 зображена схематична структура Helix-архітектури.

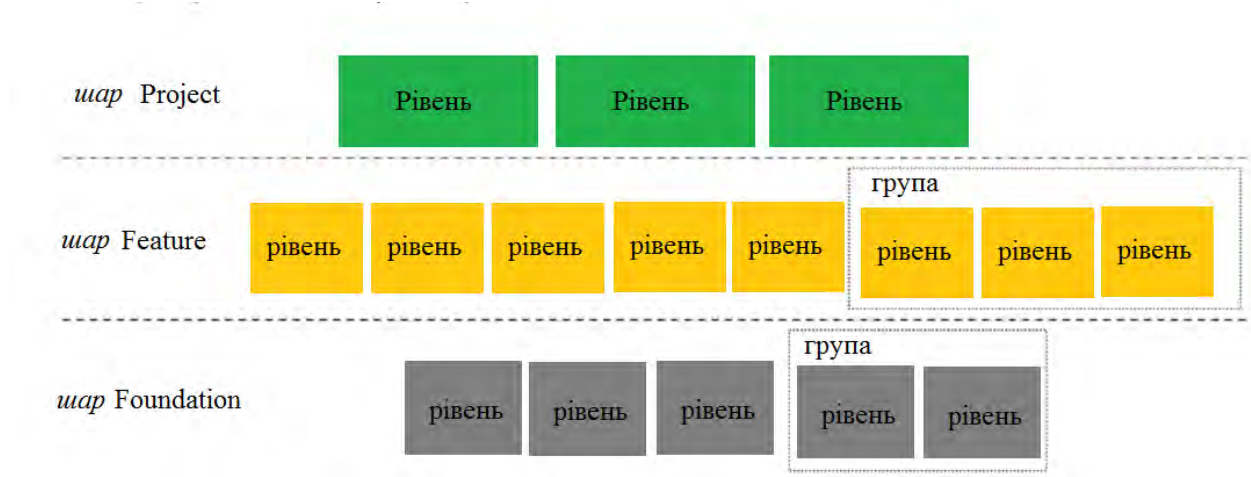


Рисунок 2.5 – Структура Helix-архітектури

2.3 Функціональні вимоги до програмного забезпечення

Як зазначалося в розділі 1, застосунок має бути розроблений для трьох типів користувачів:

- адміністратора;
- замовник;
- фрілансер.

Тому можна класифікувати наступні функціональні вимоги у відповідності до вище вказаного списку, а також ввести категорію – спільне або функціональність, яка не прив'язаний до ролі.

Програма повинна задовольняти наступні функції:

- авторизацію та реєстрацію у застосунку;
- швидку зміну ролі шляхом перемикання декількох кнопок;
- мати двунаправлений чат для користувачів та можливість ділитися файлами;
- можливість заповнення мов, якими володіє користувач;
- можливість заповнення держави, дати народження та інших даних при реєстрації.

Програма повинна задовольняти наступні функції для адміністратора:

- заблокувати користувача;

- проводити студ операції з елементарними таблицями;
- блокувати замовлення.

Програма повинна надавати наступні функції для фрілансера:

- можливість виставлення рейтингу після завершення замовлення
- можливість зчитування замовлення;
- проводити студ операції із нотатками до замовлення;
- проводити студ операції для навичок у профілі;
- проводити студ операції для сертифікатів у профілі;
- проводити студ операції для навичок у профілі;
- проводити студ операції для місць навчань у профілі;
- мати можливість для опису себе у профілі;
- переглядати список замовлень по фільтрах;
- можливість залишати та редагувати відгуки;
- можливість видалення та зчитування відгуків.

Програма повинна задовольняти наступні функції для замовника:

- можливість виставлення рейтингу після завершення замовлення;
- проводити студ операції із замовленням;
- проводити студ операції із нотатками до замовлення;
- обмежувати доступ до замовлення;
- проводити студ операції із відгуками у профілі;
- можливість фільтрації та пошуку фрілансерів за здатностями, рейтингом, іменем, мовою та іншим;
- можливість залишати відгук.

Відповідно було побудовано діаграми прецедентів для кожного з користувачів. На рис. 2.6 представлено діаграму прецедентів «Основна функціональність фрілансера».

Потік подій для розробленої діаграми прецеденту «Основна функціональність адміністратора».

Передумова: доступ до мережі інтернет, користувач авторизований як адміністратор, відкрита головна сторінка сайту.

Основний виконавець – адміністратора.

На рисунку 2.8 зображена діаграма прецедент «Основна функціональність замовника»

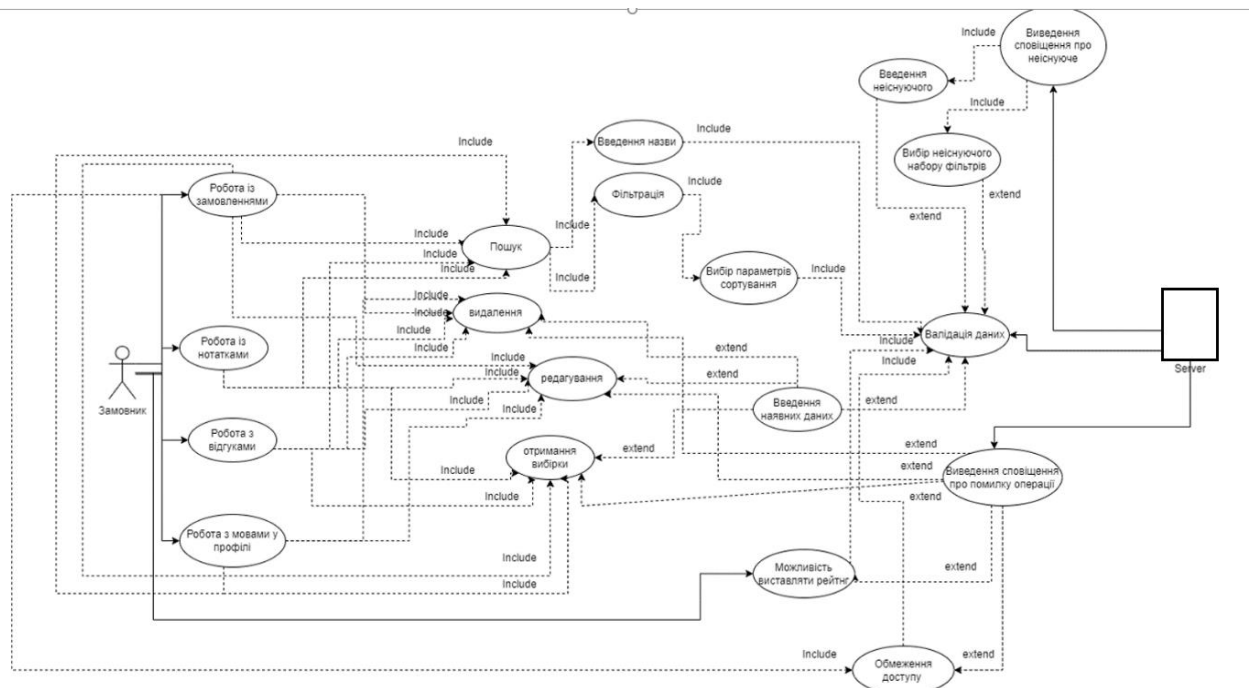


Рисунок 2.8 – Прецедент «Основна функціональність замовника»

Потік подій для розробленої діаграми прецеденту «Основна функціональність замовника».

Передумова: доступ до мережі інтернет, користувач авторизований як замовник, відкрита головна сторінка сайту.

Основний виконавець – замовник.

2.4 Проєктування структури БД

При роботі з рівнем доступу до даних буде використовуватися реляційна система управління БД MS SQL Server. Для створення самої БД прийнято рішення використовувати технологію Entity Framework та підхід Code first, при якому спочатку створюються класи доменних моделей, а також конфігурації та контекст, після чого створюється сама БД з початковими даними, які будуть знаходитися в ініціалізаторі. Це значно пришвидшить розробку замість звичайного підходу з ADO.Net. Також варто зауважити, що у поєднанні з Entity Framework буде використовуватися Linq та інтерфейси IEnumerable, IQueryable для побудови запитів до бази даних шляхом кодування, що значно спростить розуміння та підвищить читабельність коду. Також вирішено використовувати технологію Identity, яка значно спрощує розробку авторизації у застосунку та створює декілька вже готових таблиць у базі даних. Ідентифікатор має тип Guid, що робить можливим мати дуже багато записів, а також виключає можливість колізії.

Тож переходячи до структури бази даних, розглянемо сутності користувачів та допоміжних таблиць. Розпочати слід з найбільшої таблиці AspNetUser, яка уособлює у собі основні дані, які буде містити кожен користувач незалежно від ролі, це також дає можливість зміни ролі, що буде значно зменшувати кількість записів у базі даних та буде зменшувати кількість кроків для реєстрації на іншу роль. Цей тип є початковим і він розширюється власними полями. На рисунку 2.9 зображені поля та їх типи, а також зазначається, чи можуть вони мати порожнє значення.

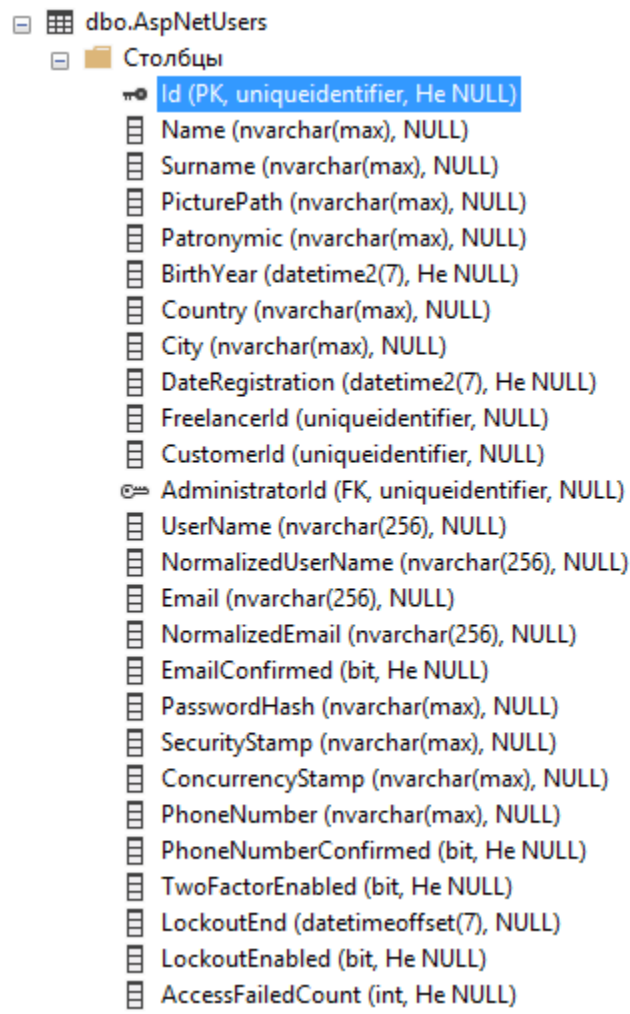


Рисунок 2.9 – Детальна типізація сутності AspNetUser

Наступною є сутність Customer. Це розширення, яка має на меті додати дані, які потрібні для функціонування користувача. На рисунку 2.10 зображені поля та їх типи, а також зазначається, чи можуть вони мати порожнє значення.

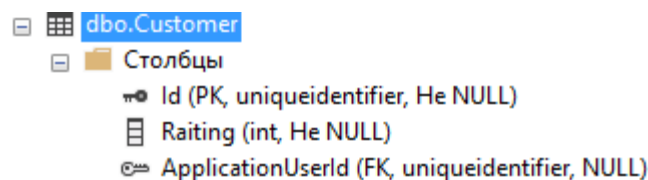


Рисунок 2.10 – Детальна типізація сутності Customer

Наступною є сутність Administrator, яка має на меті розширення для функціонування користувача. У майбутньому ця сутність може бути

розширена. На рисунку 2.11 зображені поля та їх типи, а також зазначається, чи можуть вони мати порожнє значення.

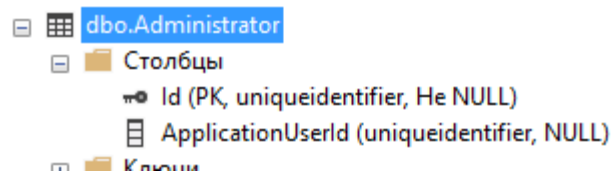


Рисунок 2.11 – Детальна типізація сутності Administrator

Наступною є сутність Freelancer, яка має на меті розширення для функціонування користувача. У майбутньому ця сутність може бути розширена. На рисунку 2.12 зображені поля та їх типи, а також зазначається, чи можуть вони мати порожнє значення.

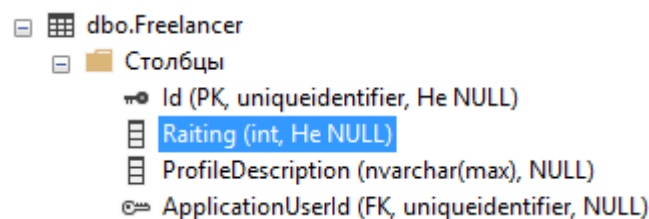


Рисунок 2.12 – Детальна типізація сутності Freelancer

Наступними подані допоміжні таблиці, які були створені за участі технології Identity, такими сутностями виступають таблиці:

- ASPNetRoles;
- ASPNetRoleClaims;
- ASPNetUserLogins;
- ASPNetUserTokens.

Ці сутності, наявні у документації технології.

Далі представлені допоміжні таблиці, які мають лише поле імені. Вони були створені для того, щоб скоротити об'єм пам'яті та мати у таблиці лише ключі. Такими сутностями виступають таблиці:

- Country;

- SubCategory;
- SocialLink;
- SkillLevel;
- Language;
- Degree;
- Categories;
- Availability;

Далі на рисунку 2.13 зображена структура типів, яка буде загальною для усіх вище вказаних таблиць з полем ім'я.

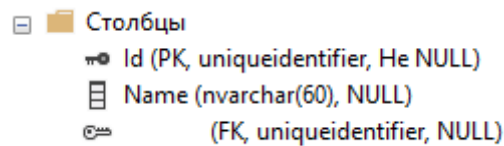


Рисунок 2.13 – Детальна типізація сутності з одним полем ім'я

Далі розглянемо сутність Skill, яка розширює дані Фрілансера та має наступні поля, які зображені на рисунку 2.14. Ця сутність потрібна для профілю фрілансу.

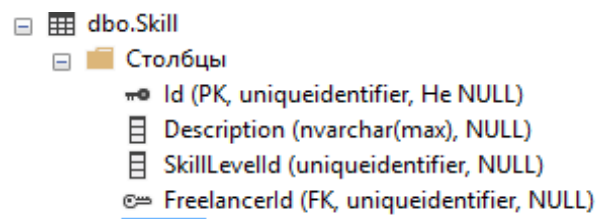


Рисунок 2.14 – Детальна типізація сутності Skill

Наступна сутність – це Order. Це сутність, яку може створювати Замовник для створення контракту на виконання послуг з фрілансером. Далі на рисунку 2.15 зображена сутність Order з усіма наявними типами.

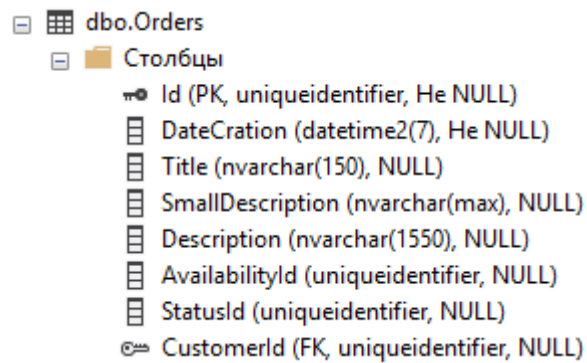


Рисунок 2.15 – Детальна типізація сутності Order

Далі розглянемо сутність Note. Ця сутність потрібна для заміток у замовленні. Детальна структуризація цієї сутності зображена на рисунку 2.16.

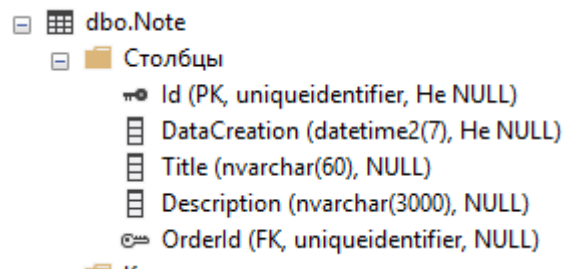


Рисунок 2.16 – Детальна типізація сутності Note

Наступна сутність – це FilePath, яка потрібна для доступу до файлів у файловому провіднику. Детальна структуризація цієї сутності зображена на рисунку 2.17.

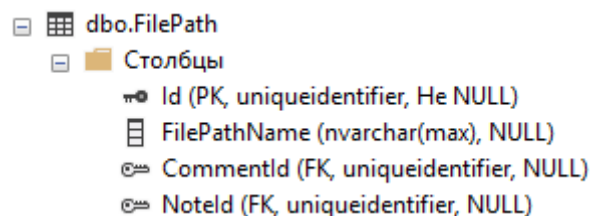


Рисунок 2.17 – Детальна типізація сутності FilePath

Сутність Feedback можна групувати у підрозділ Профіль. Вона потрібна для того, щоб зберігати відгуки від користувачів та виводити їх на профіль. Детальна структуризація цієї сутності зображена на рисунку 2.18.

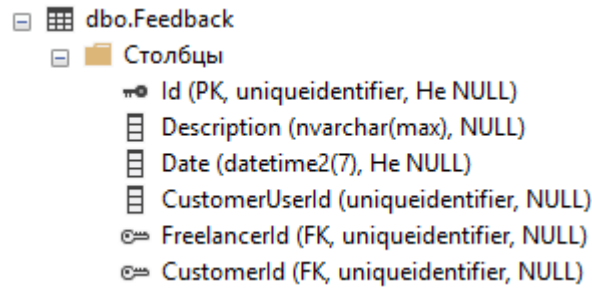


Рисунок 2.18 – Детальна типізація сутності Feedback

Сутність Education теж входить до групи профілю. Та потрібно для збереження інформації про навчання фрілансера. Детальна структуризація цієї сутності зображена на рисунку 2.19.



Рисунок 2.19 – Детальна типізація сутності Education

Сутність Certificate теж входить до групи профілю. Потрібна для збереження інформації про сертифікати, які здобув фрілансер. Детальна структуризація цієї сутності зображена на рисунку 2.20.

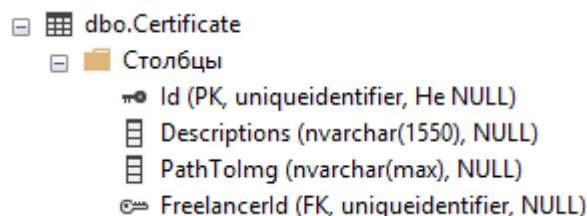


Рисунок 2.20 – Детальна типізація сутності Certificate

Наступна сутність – це сутність Chat. Ця група відповідає за двонаправлену комунікацію між сутностями замовника та фрілансера. Детальна структуризація цієї сутності зображена на рисунку 2.21.

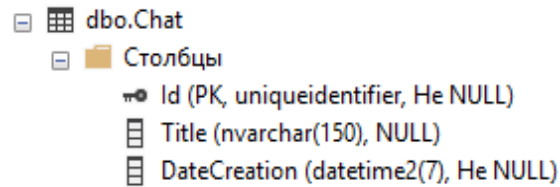


Рисунок 2.21 – Детальна типізація сутності Order

Сутність Comments входить до групи комунікації. Це найменша одиниця чату. Детальна структуризація цієї сутності зображена на рисунку 2.22.

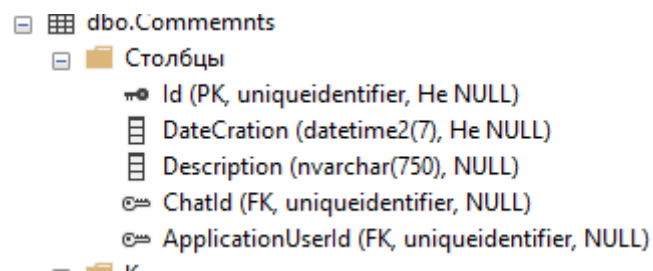


Рисунок 2.22 – Детальна типізація сутності Comments

Повна схема бази даних зображена на рисунку 2.23.

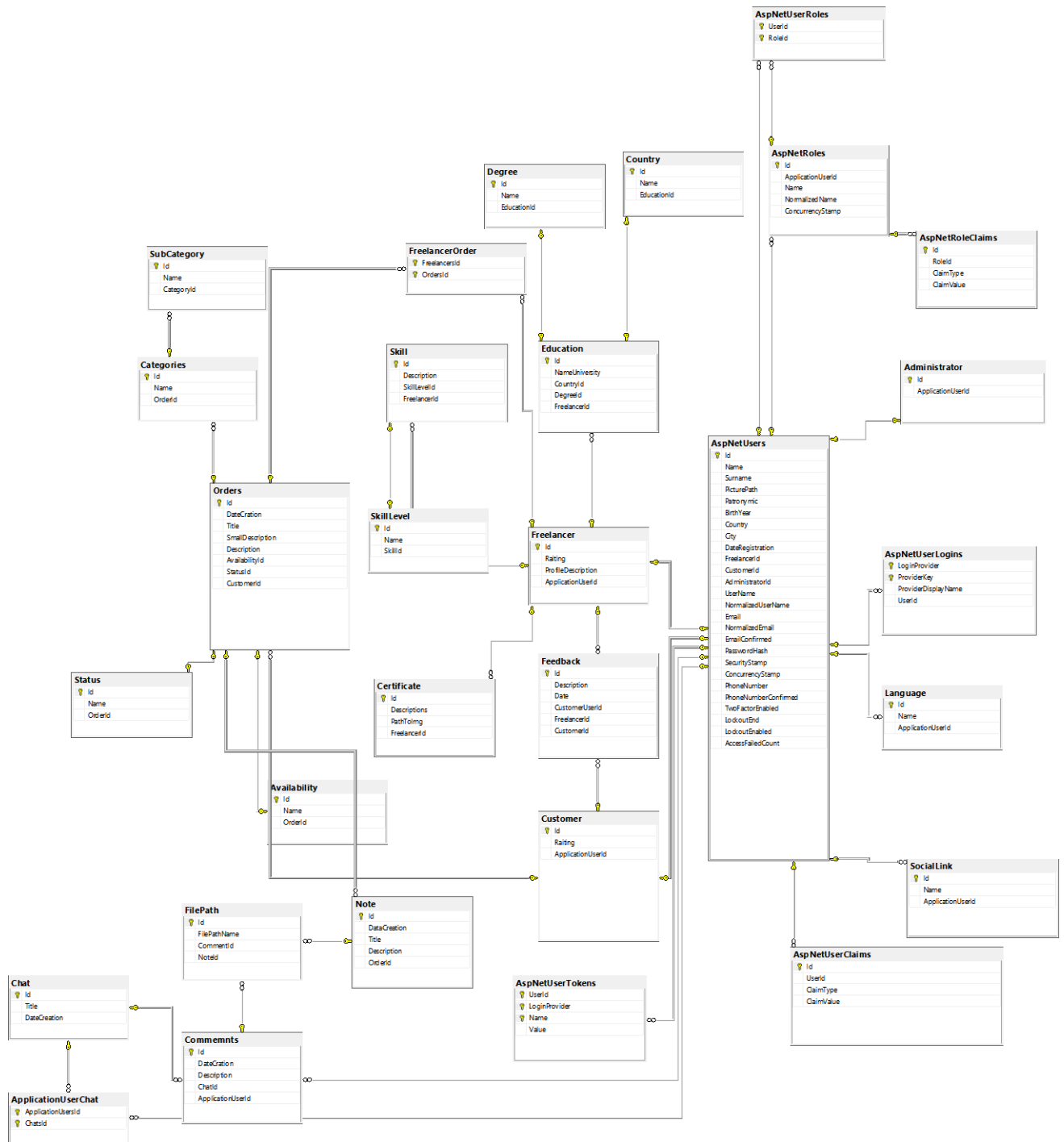


Рисунок 2.23 – Повна структура бази даних

2.5 Структура програмного проєкту

Як вже було обговорено, система має бути побудована на багатошаровій архітектурі. В ідеях архітектури та структури програмного застосунку слід пригадати про принципи, які гарантують у подальшому можливості до масштабування та оптимізації, які безумовно ляжуть у розробку застосунку:

- Solid;
- DI (Dependency Injection);
- IoC (Inversion of Control);
- Kiss.

Першим рівнем є рівень даних, який у себе буде включати наступні паттерни: Repository, Domain model, UnitOfWork, Factory. На цьому рівні буде виконуватися уся робота з даними. А репозиторій буде універсальним для перевикористання його функціональності. На рисунку 2.24 наведена структура папок та файлів першого рівня.

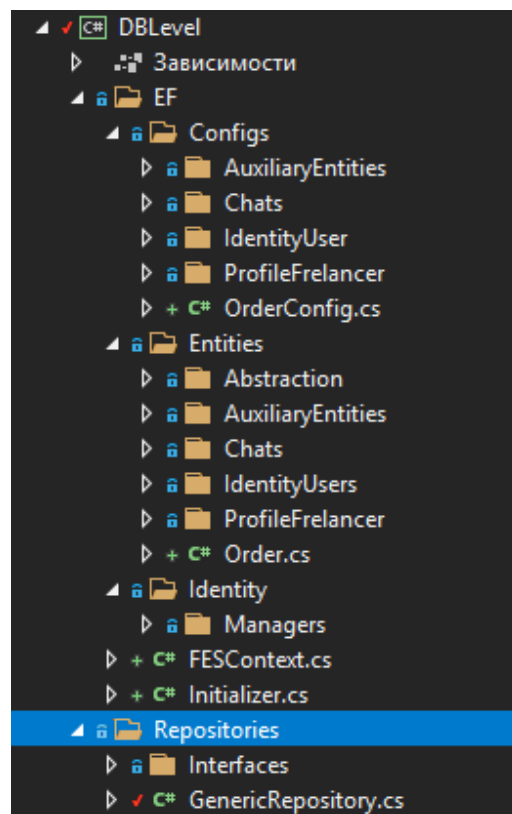


Рисунок 2.24 – Структура папок та файлів першого рівня

Другим рівнем буде виступати рівень бізнес логіки, що уособлює в собі патерни: DTO, Service. Цей рівень буде оброблювати дані та віддавати їх далі.

Третім рівнем буде виступати рівень представлення, в якому будуть зосередженні файли представлень інтерфейсу, що будуть оброблюватися

контролерами. На рисунку 2.25 наведена структура папок та файлів другого рівня.

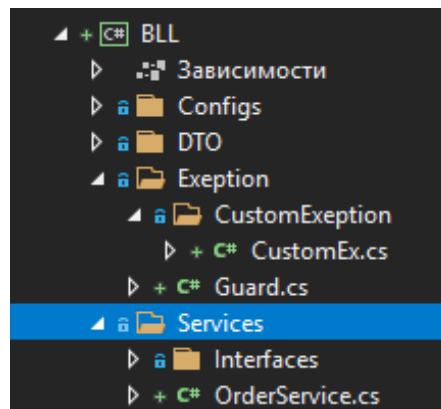


Рисунок 2.25 – Структура папок та файлів другого рівня

Слід зазначити, що має використовуватися модульне тестування для більш надійної роботи програми у випадку зміни коду з боку розробника (використовуватиметься підхід TDD). На рисунку 2.26 наведена структура папок та файлів третього рівня.

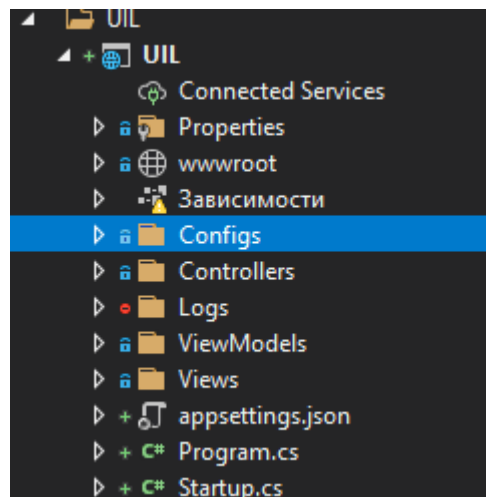


Рисунок 2.26 – Структура папок та файлів третього рівня

До кожного з проєктів мають бути додані проєкти з тестами, які будуть повністю повторювати структуру проєктів, під які вони створюються.

2.6 Вимоги до програмного забезпечення

Виходячи з всього, що описано було вище, можна сформулювати вимоги до програмного забезпечення. Почати потрібно з основної мови програмування, а саме для серверної частини – це C#. Платформою, що має використовуватися для розробки програмного застосунку, є WEB API (вона є універсальною для використання). Версія проєктів повинна бути .NET Core 3.1 Архітектура програмного застосунку має бути багатошаровою та складатися з 3 шарів, які вже були описані раніше. Потрібно чітко слідувати пункту 2.5, у якому вказані патерни, які потрібно використати. Для створення модульних тестів потрібно створити окремі проєкти відповідно до шарів застосунку та використовувати Xunit, Moq, AutoFixture для їх створення. Вимоги до бази даних описані у пункті 2.4. Також потрібно документувати за допомогою XML коментарів створений код, передбачити обробку помилок та виключних ситуацій, впевнитися у працездатності застосунку.

2.7 Висновки за другим розділом

У даному розділі були здійснені початкові кроки в процесі створення застосунку: проаналізовані варіанти з вибору архітектур та вибрана архітектура, що найбільше відповідає критеріям поставленого завдання, – багатошарова, була визначена структура бази даних та підхід до її створення, розглянуті сучасні мови програмування та вибрана мова C#, були визначені основні патерни та технології, які мають бути застосовані під час розробки.

3 РОЗРОБКА ПРОГРАМИ

В даному розділі буде розглянуто алгоритм конвеєру запитів, описані та пояснені основні модулі, рівні та класи.

3.1 Алгоритм функціонування програмного забезпечення

При потраплянні користувача на головну сторінку він повинен бачити вікно, у якому будуть розташовуватися привітання та надаватися можливість користувачу авторизуватися чи зареєструватися. Після проходження пункту авторизації користувач потрапляє у інтерфейс, де він бачить наявну йому функціональність.

Для фрілансера надаються:

- можливість дивитися замовлення, за які він відповідальний;
- можливість роботи з власним профілем;
- можливість роботи з чатом.

Для адміністратора надаються:

- можливість роботи з елементарними таблицями.
- можливість роботи з чатом.

Для замовника надаються:

- можливість роботи із замовленнями;
- можливість вибору та пошуку фрілансерів;
- можливість роботи з власним профілем;
- можливість роботи з чатом.

Програма розроблена на Web API застосунку, тому у програмі використовується звичайний конвеєр запитів, який є у цьому застосунку за замовченням. Від користувача на сервер потрапляє запит, який потім проходить наступні етапи:

- Middlewares (служби);

- Routing (адресація до контролеру та його дії);
- Controller Initialization (ініціалізація контролерів та допоміжних класів);
- Controller action execution (виклик коду із параметрами);
- Result Execution (результат роботи коду);
- View Rendering (обробка результату у представленнях).

Схематично це зображено на рисунку 3.1.

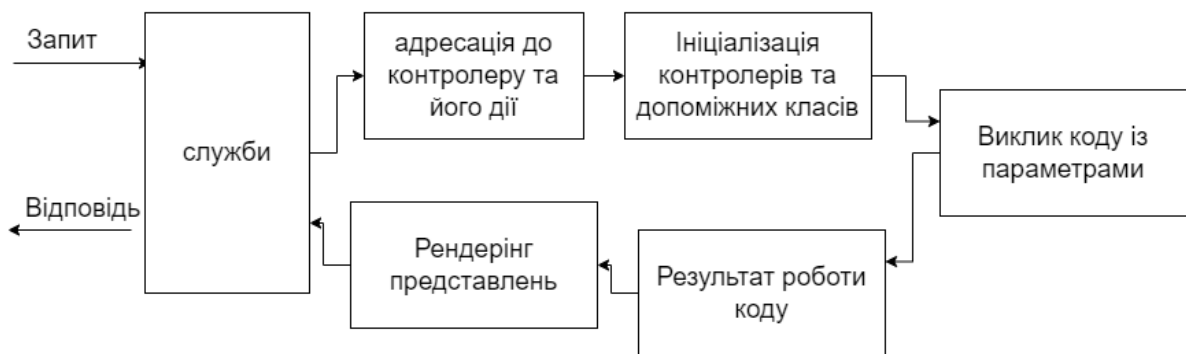


Рисунок 3.1 – Схематичне зображення проходження одного процесу через застосунок

Middlewares (служби) – це блоки, з яких будується HTTP пайплайн. Кожен з таких блоків приймає HTTP контекст та може його модифікувати чи якось обробити, після чого він передає чи не передає його далі. Таких блоків у програмі може бути дуже велика кількість. HTTP контекст має у собі величезну кількість даних про HTTP запит, що дозволяє повернути користувачу повну відповідь.

Далі виконується етап маршрутизації. В ньому із URL витягуються усі необхідні параметри і передаються далі в контролер. Для роботи самого роутингу наразі використовуються дві middlewares:

- UseRouting;
- UseEndpoints.

Після обробки рядка запиту программа виконує ініціалізацію потрібного контролеру, сервісів та репозиторіїв, які потрібні для обробки запиту за допомогою класів `ResourceInvoker` та `ControllerActionInvoker`.

Після чого буде отримано результат, який потім оброблюється службою `Razor` і відправляється користувачу результат у вигляді сторінок у браузері з даними.

Далі слід розглянути бізнес процес отримання замовником даних із серверу. Цей процес наочно демонструє ще раз роботу конвеєра запитів. Бізнес процес зображений на рисунку 3.2.

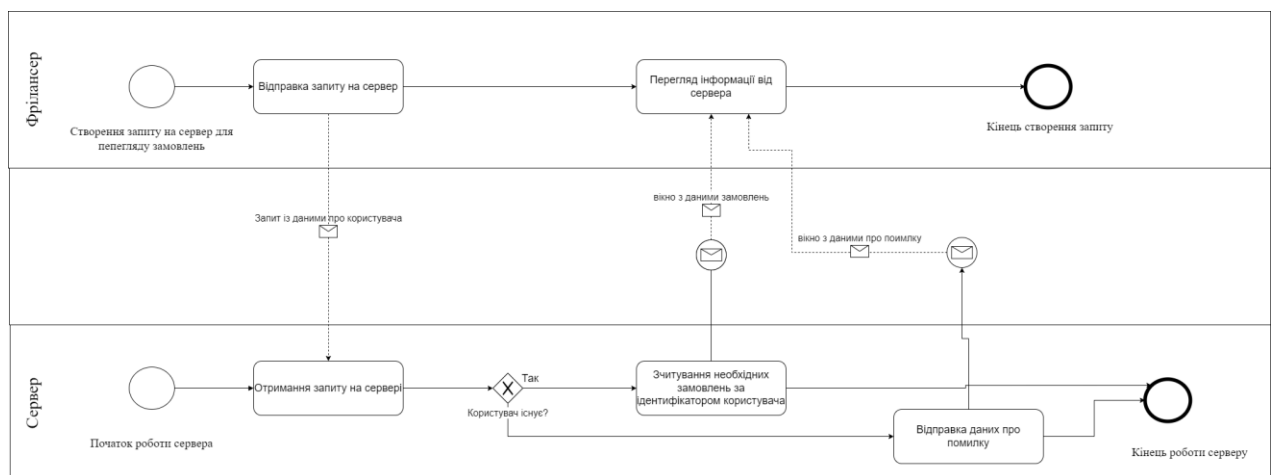


Рисунок 3.2 – Бізнес процес конвеєра запитів

3.2 Опис модулів реалізованої програми

Структура проєкту поділяється на три модулі:

- DAL;
- BLL;
- UIL.

Для розбору першого рівня, DAL рівня даних, потрібно розглянути наступні рівні:

- Entity Framework;
- UnitOfWork;

- розширення;
- репозиторії.

Окремо потрібно виділити патерн UnitOfWork, який у собі уособлює взаємодію з усіма репозиторіями. Саме через нього здійснюється взаємодія з усім рівнем даних та з самою базою даних за допомогою репозиторіїв. Також він має метод `SaveChangesAsync`, що відповідає за збереження даних у БД. Метод `Dispose` використовується для вивільнення пам'яті об'єкту `IdentityDbContext`. `IdentityDbContext` виступає у ролі некерованих ресурсів, тому збирач сміття не зможе вивільнити пам'ять. Тому в силу вступає інтерфейс `IDisposable`, що допомагає у цьому. Схематичне зображення полів та методів класу наведено на рисунку 3.3.

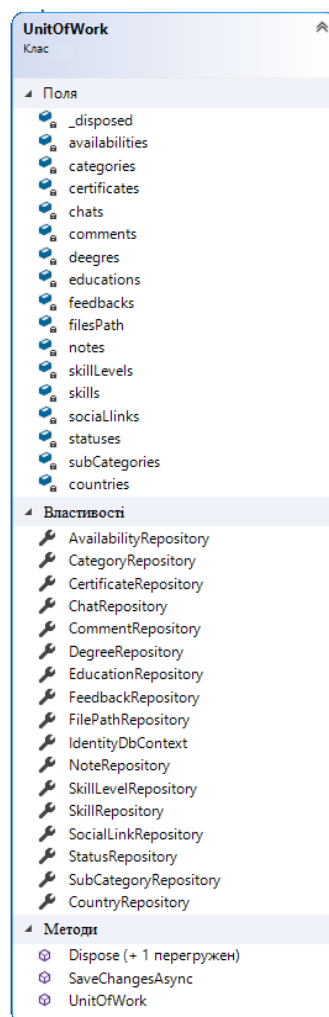


Рисунок 3.3 – Схематичне зображення полів та методів UnitOfWork

На рівні Entity Framework знаходяться конфігурації до самих моделей, у яких знаходяться зв'язки до сутностей, а також відповідні типи у базі даних. У функції Configure проводиться робота по конфігурації цієї сутності у БД. Ці файли розрізняються лише налаштуваннями у методі Configure. Приклад схеми наведено на рисунку 3.4 для класу CategoryConfig.

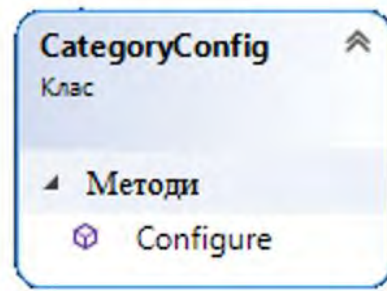


Рисунок 3.4 – Схематичне зображення полів та методів класу CategoryConfig

Також на цьому рівні знаходяться доменні моделі, які у собі містять дані щодо властивостей, зв'язків моделей. Абстрактною моделлю виступає клас BaseEntity, який потрібен для економії часу і який у собі має лише ідентифікатор. На рисунку 3.5 зображено його схематичне зображення.

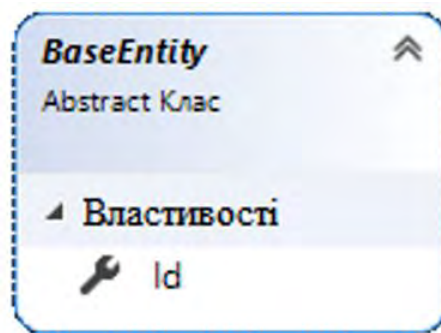


Рисунок 3.5 – Схематичне зображення полів та методів класу BaseEntity

Далі на рисунку 3.6 буде наведена група допоміжних класів, що потрібні для початкової ініціалізації даних, а також наведений клас Note, який потрібен для зберігання додаткової інформації для замовлень.

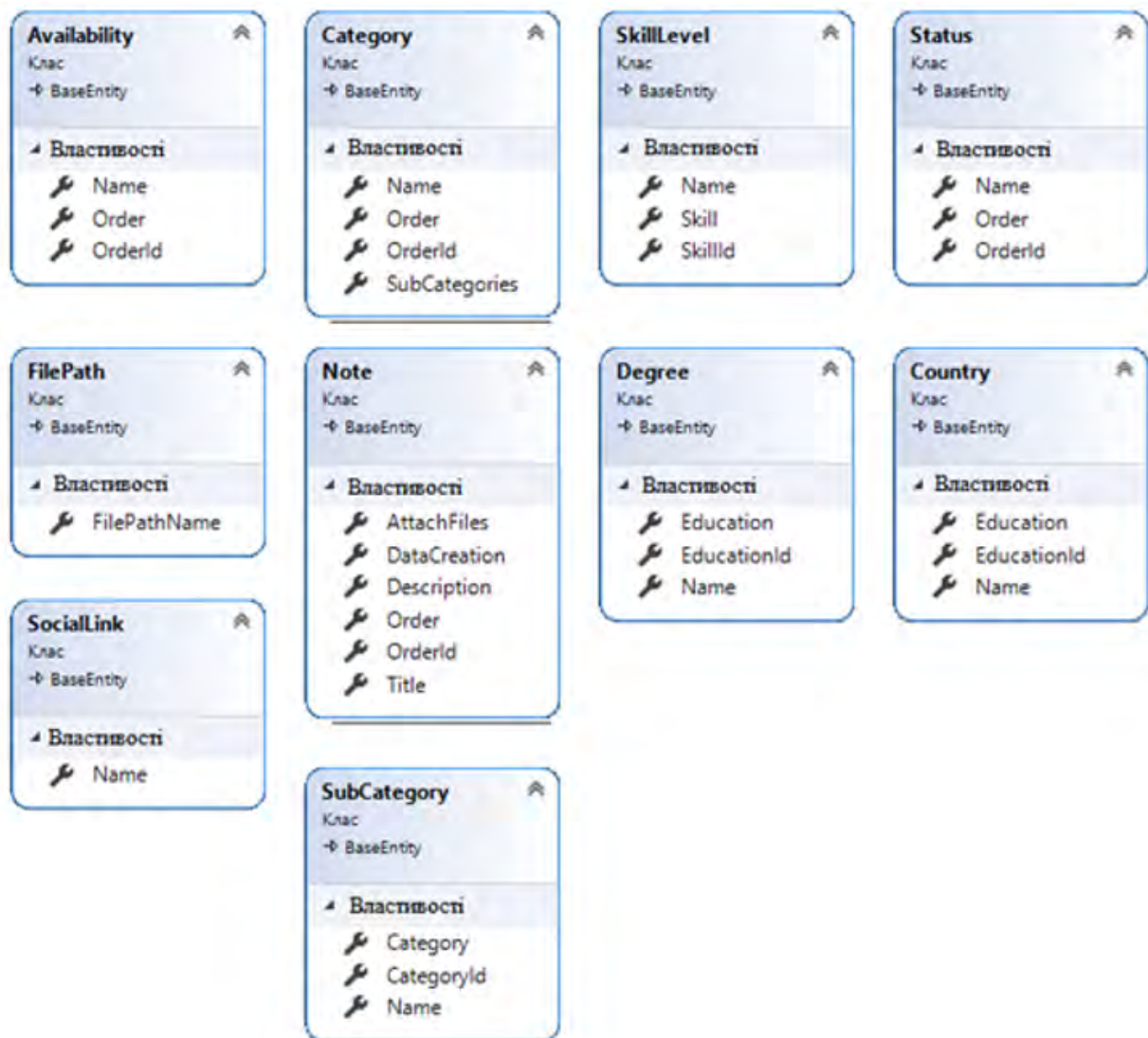


Рисунок 3.6 – Схематичне зображення полів та методів класів групи допоміжних класів

Окремою групою класів будуть виступати класи Comment та Chat, які будуть входити до групи чату. Далі буде наведений рисунок 3.7 з відповідною схемою цих класів.

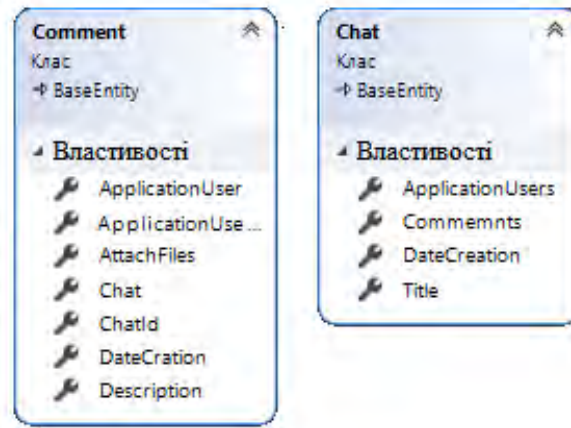


Рисунок 3.7 – Схематичне зображення полів та методів класів групи класів чату

Далі наводиться група класів користувачів, у якій головним класом є ApplicationUser, який у собі містить основні дані щодо кожного користувача. Також у цій групі наведені класи Customer, Freelancer, Administrator, що виступають доповненнями до даних класу ApplicationUser. Окремо наявна сутність ApplicationRole, що виступає у ролі даних для визначенням ролі користувача. Відповідні схеми до цих сутностей наявні на рисунку 3.8.

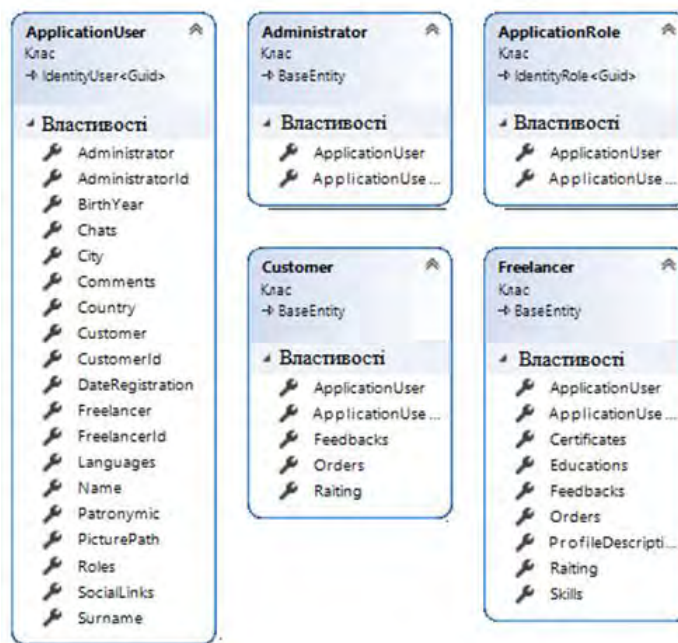


Рисунок 3.8 – Схематичне зображення полів та методів класів групи класів користувачів

Наступною групою класів є профіль Фрілансера, який заслуговує окремої уваги. Він має додаткові чотири сутності які будуть додавати профілю фрілансера додаткову інформацію. Відповідні схеми до цих сутностей наявні на рисунку 3.9.



Рисунок 3.9 – Схематичне зображення полів та методів класів групи класів профілю Фрілансера

Окремою сутністю виступає клас Order. Відповідна схема зображена на рисунку 3.10.

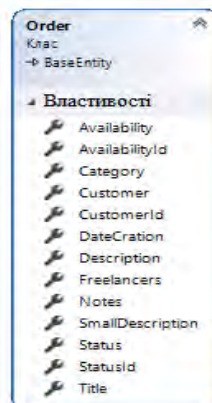


Рисунок 3.10 – Схематичне зображення полів та методів класу Order

Найголовнішим класом у рівні Entity Framework виступає клас FESContext, який визначає контекст даних, через який програма буде взаємодіяти БД. Кожна з DbSet колекцій має набір сутностей, через який можна звертатися до БД із запитом. Також у класу є метод OnModelCreating, який потрібен для конфігурації сутностей. Відповідна схема зображена на рисунку 3.11.



Рисунок 3.11 – Схематичне зображення полів та методів класу FESContext

Також у додатку є клас_INITIALIZER, який виступає сховищем початкових даних при генерації БД. У випадку застосування – елементарних таблиць. Відповідна схема зображена на рисунку 3.12.

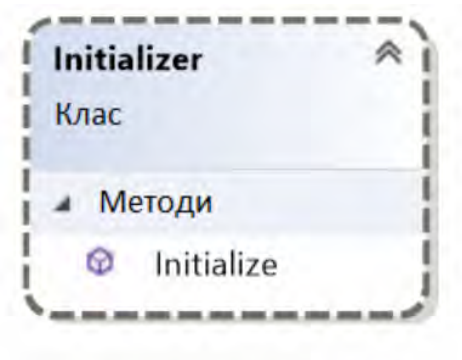


Рисунок 3.12 – Схематичне зображення полів та методів класу Initializer

Далі слід розглянути рівень репозиторіїв. Найстаршим класом в ієрархії виступає `GenericRepository`. Він уособлює у собі базовий набір методів, який потрібен для взаємодії з кожною сутністю. Далі цей репозиторій наслідують інші у відповідності з кожною сутністю. Така архітектура репозиторіїв дозволяє значно структурувати та економити час на написання коду. Схематичне зображення такої ієрархії зображено на рисунку 3.13.

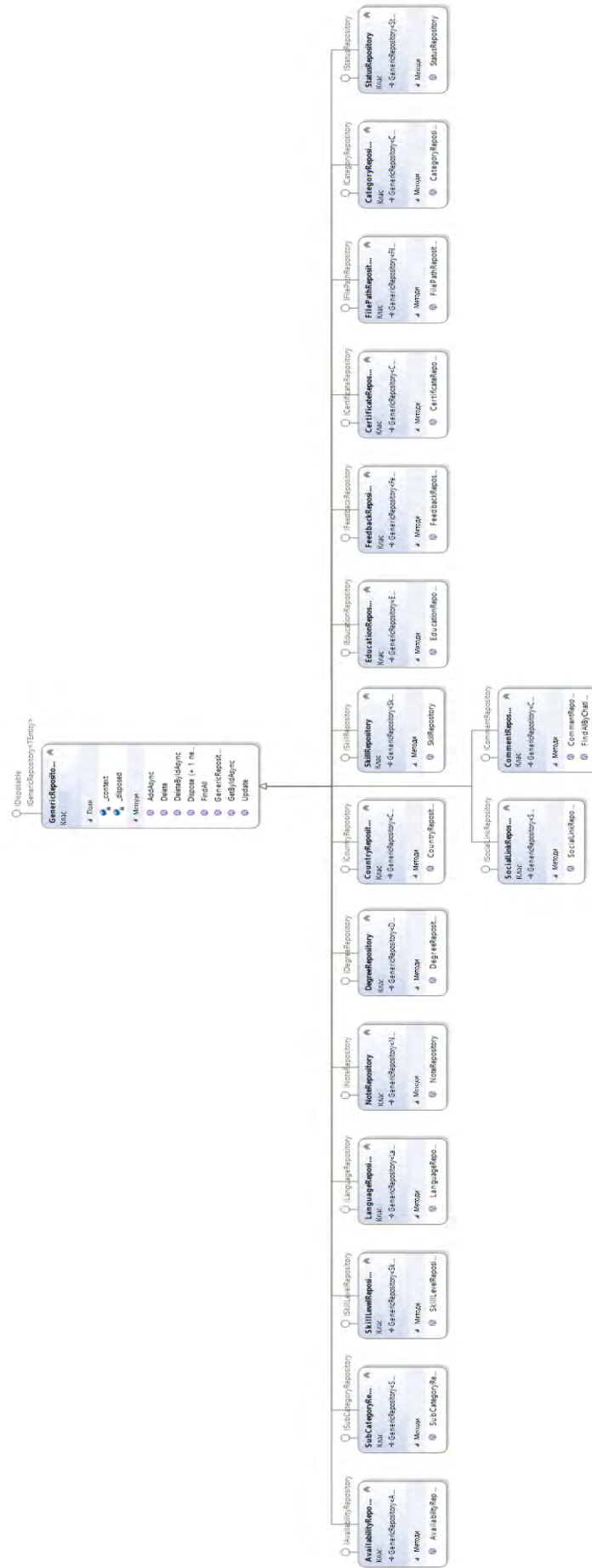


Рисунок 3.13 – Схематичне зображення ієрархії репозиторіїв

Останнім рівнем, який залишився, є рівень розширень. Цей рівень потрібен для роботи наступних рівнів. Він уособлює у собі два класи `DateTimeProvider` та `IdGenerator`. Вони потрібні для коректного створення сутностей на рівні бізнес логіки. Відповідні схеми представлені на рисунку 3.14.

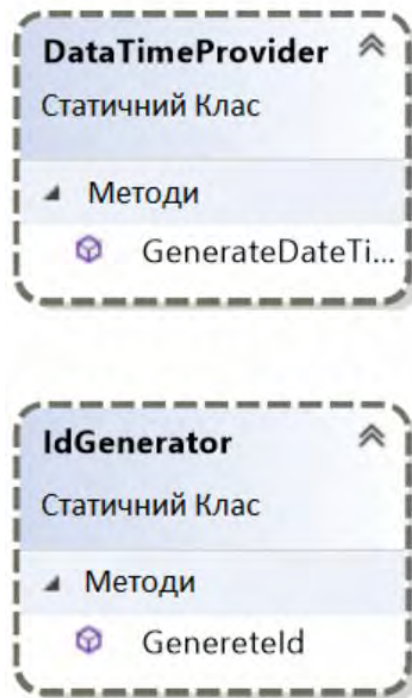


Рисунок 3.14 – Схематичне зображення методів та полів класів групи розширень

Для представлення другого рівня BLL рівня бізнес логіки потрібно розглянути наступні рівні:

- DTO;
- Exeption;
- Services;
- Configs.

На рівні сервісів знаходяться чотири сервіси, які імплементують логіку, потрібну для crud операцій з даними. В них агрегуються інтерфейси

IUnitOfWork та IMapper. Потім на їх місця підставляються зареєстровані реалізації цих класів і можна використовувати їх для своїх потреб.

На рисунку 3.15 зображено групу сервісів, які потрібні для функціонування чату.

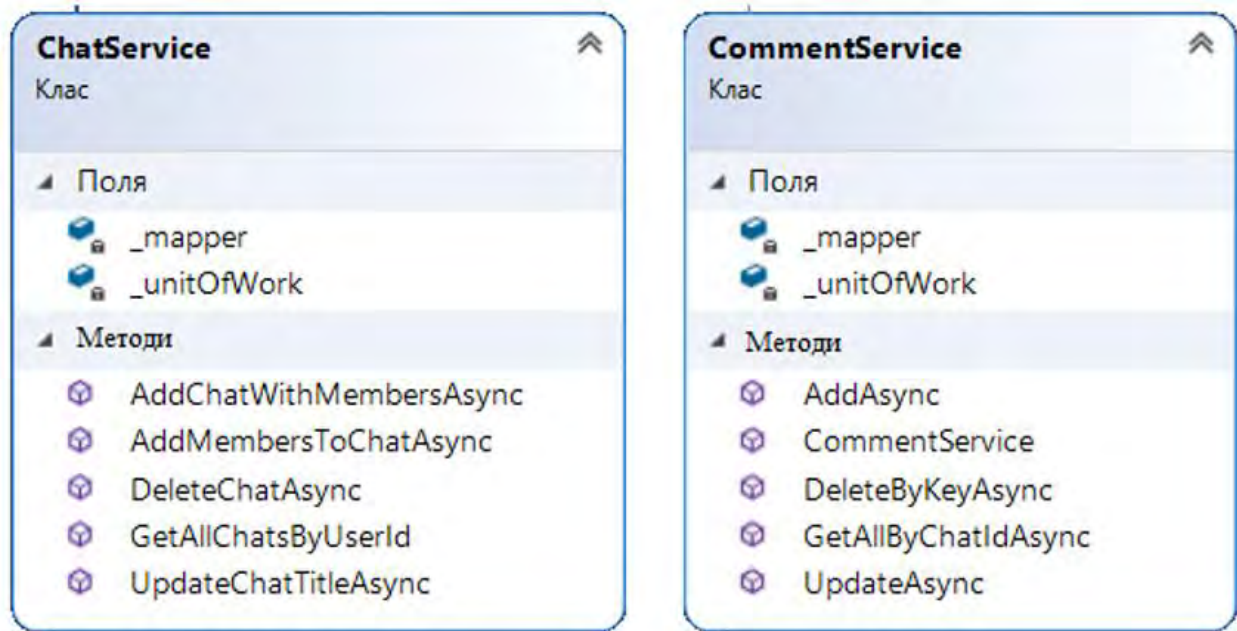


Рисунок 3.15 – Схематичне зображення методів та полів класів групи чату сервісів

На рисунку 3.16 зображено клас, який потрібен для функціонування замовлень.

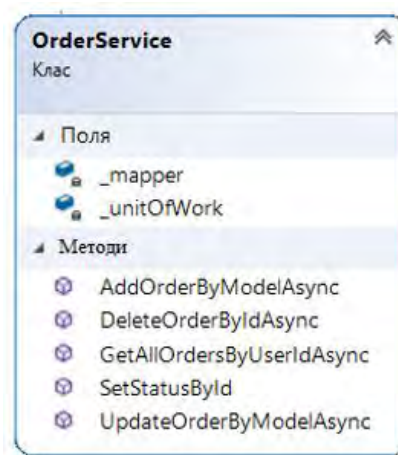


Рисунок 3.16 – Схематичне зображення методів та полів класу OrderService

На рисунку 3.18 зображено клас, який потрібен для функціонування адміністративної панелі.

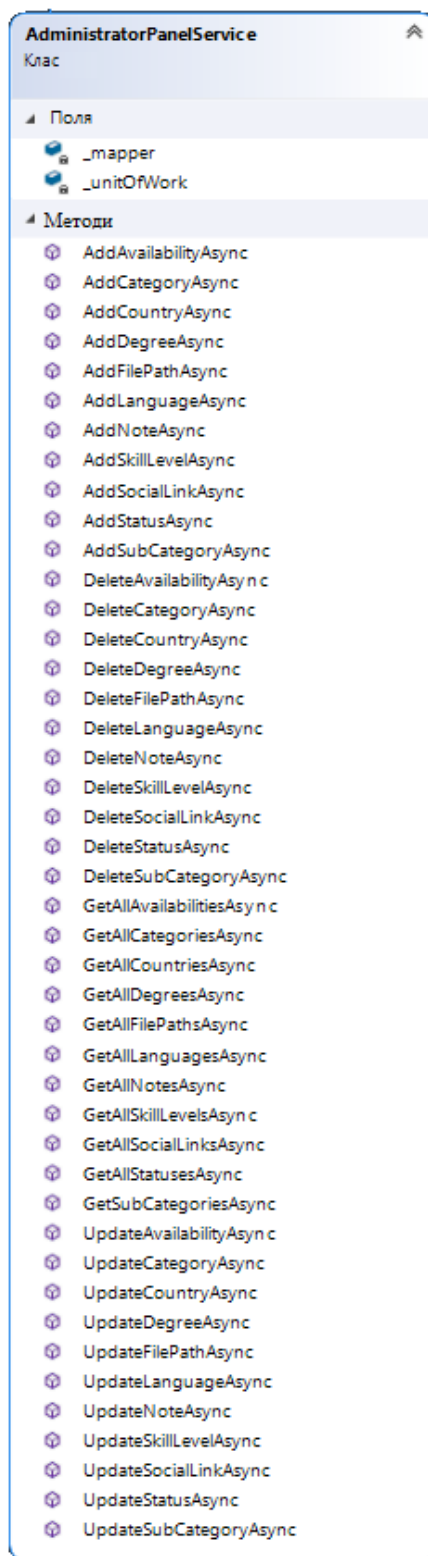


Рисунок 3.18 – Схематичне зображення методів та полів класу
AdministrativePanelService

На рисунку 3.17 зображено клас, який потрібен для функціонування профілю замовника та фрілансера.

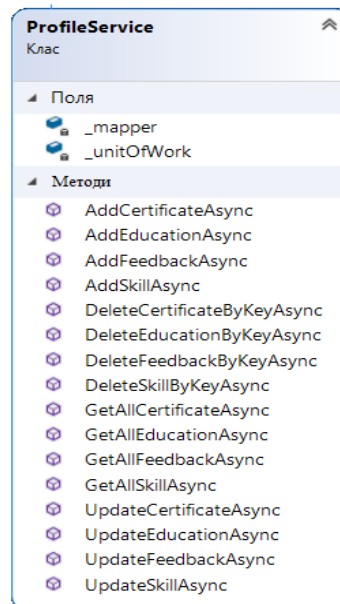


Рисунок 3.17 – Схематичне зображення методів та полів класу ProfileService

На наступному рівні Exception знаходиться клас Guard та користувацькі виключення. На рисунку 3.19 зображено схему класу Guard.

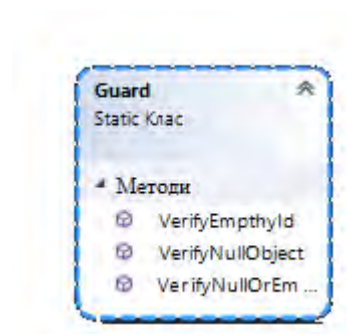


Рисунок 3.19 – Схематичне зображення методів та полів класу
AdministrativePanelService

На наступному рівні знаходяться DTO, які потрібні для пагінації доменних моделей. На рисунку 3.20 наводяться 2 DTO коментарів. На їх

прикладі можна побачити, що з моделі Comment витягнуті лише деякі дані, які будуть потрібні у майбутньому для обробки.

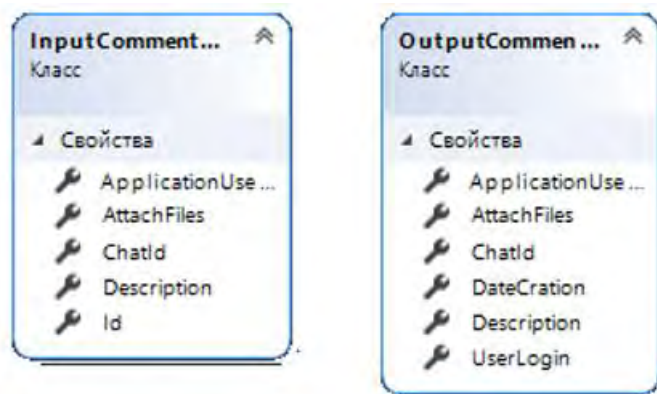


Рисунок 3.20 – Схематичне зображення методів та полів класів InputCommentDTO та OutputCommentDTO.

Окремим рівнем виступає рівень конфігурації, у якому лежить конфігурація AutoMapper, яка потрібна для переносу даних з однієї моделі до іншої. Схематична конфігурація класу AutoMapper зображена на рисунку 3.21

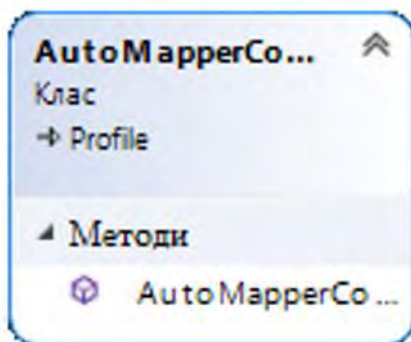


Рисунок 3.21 – Схематичне зображення методів та полів класу AutoMapper

На третьому шарі UIЛ знаходяться наступні рівні:

- конфігурації;
- контролери;
- служби;
- ViewModels;

– Views.

На рівні конфігурацій знаходяться 2 файли конфігурацій AutoMapperConfig та MappingProfile. Вони потрібні для роботи AutoMapper. Схематичні конфігурації цих файлів представлені на рисунку 3.22.

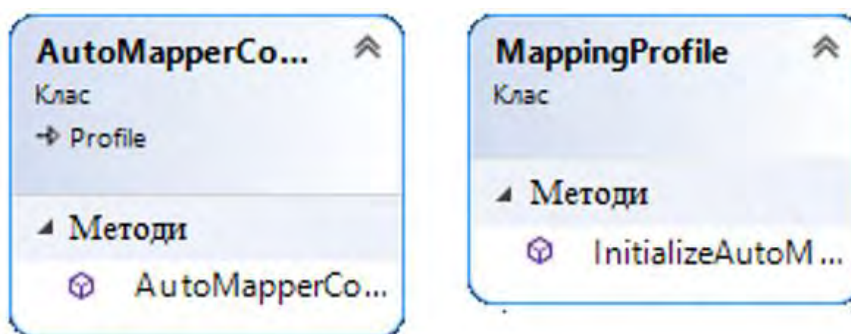


Рисунок 3.22 – Схематичне зображення методів та полів класів MappingProfile, AutoMapperConfigUIL.

Окремо потрібно виділити класи Program та Startup. Клас Program потрібний для запуску застосунку та має у собі конфігураційні налаштування. Також у цьому класі ініціалізується логер та контекст. У класі Startup проводяться початкові реєстрації залежностей між класами та інтерфейсами, а також саме цей клас містить пайплайн. Сигнатури цих класів наводяться нижче на рисунку 3.23.

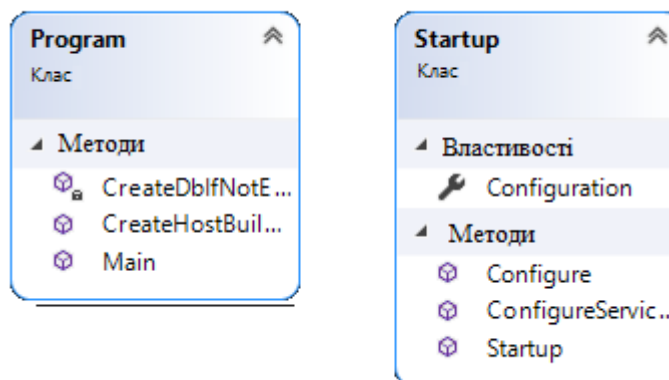


Рисунок 3.23 – Схематичне зображення методів та полів класів Program, Startup.

На рівні служб знаходиться глобальний оброблювач помилок GlobalLoggerMiddleware. У цьому класі відбувається обробка помилок, а також міститься запис, що запит був успішний та відбувається переадресація його далі по службам. Схематично цей клас зображений на рисунку 3.24.

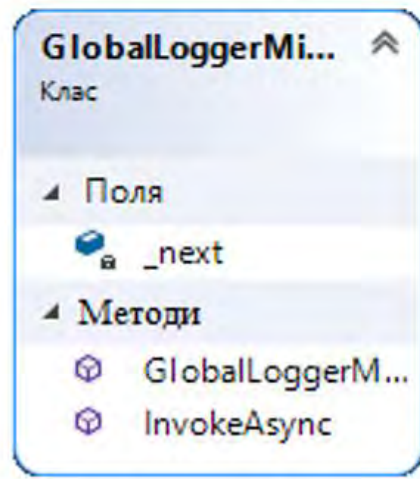


Рисунок 3.24 – Схематичне зображення методів та полів класу GlobalLoggerMiddleware

На рівні View знаходяться представлення, які представляють із себе звичайні HTML-файли з елементами C#, який генерує розмітку.

На рівні ViewModels знаходяться класи, які потрібні для використання у представленнях. Вони передають дані до Razor-сторінок.

На рівні контролерів знаходиться маршрути з відповідними функціями, які викликають відповідні сервіси та проводять операції з даними, після чого надають користувачу відповідь.

3.3 Висновки за розділом 3

В розділі було розглянуто алгоритм роботи пайплайну, який виконує основну роботу, даючи відповіді на запити користувачів. Були розглянуті по модулям та групам класи, які потрібні для коректної обробки запиту

користувача та надання йому відповідних даних. Було в цілому розглянуто серверний додаток.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Даний клієнтський застосунок розроблений для аудиторії людей, які будуть як потребувати, так і надавати послуги. Глобально програма має на меті виконувати регулювання відносин між фрілансерами та замовниками. В якості додаткової регулюючої сторони виступає адміністратор, який стежить за актуальністю даних на платформі. До ролей замовника та фрілансера може долучитися будь-який бажаючий. Адміністратори ж мають створюватися окремо за допомогою застосунків по типу Microsoft Management Studio, бо у програмі на теперішній момент ця функціональність не передбачена.

Із додаванням категорій даний клієнтський застосунок може бути задіяним у дуже широкому колі фрілансової біржі.

4.2 Умови виконання програми

Оскільки програма написана на версії Net Core 3.1, даний клієнтський застосунок може працювати на Windows, Linux та macOS.

Умовами виконання програми є наявність усіх необхідних модулів, пакетів та залежностей, які потрібні для функціонування програмного застосунку. Клієнтом програмного застосунку виступає браузер. Бажано використовувати сучасний браузер із підтримкою виходу в інтернет.

Апаратні вимоги залежать перш за все від кількості клієнтських запитів, а також кількості запущених застосунків. Місце на жорсткому диску залежить від кількості файлів, які будуть зберігатися на сервері, а також від розмірів бази даних. Мінімальні апаратні вимогами саме до серверної частини будуть наступні:

- оперативна пам'ять: від 8 Гб;
- місце на жорсткому диску під програму: 8 Гб;

– процесор повинен бути Intel чи AMD старше від I5 третього покоління або AMD Ryzen 5 5600H та старше.

Оскільки серверний додаток не потребує засобів візуалізації, то вони не будуть зазначені у апаратних вимогах.

Для розгортання серверу можна використовувати будь який з вебсерверів для Windows або інших операційних систем.

4.3 Виконання програми

Перед виконанням програми слід запустити серверний застосунок. Після чого для оператора відкриється головне вікно, яке зображено на рисунку 4.1.

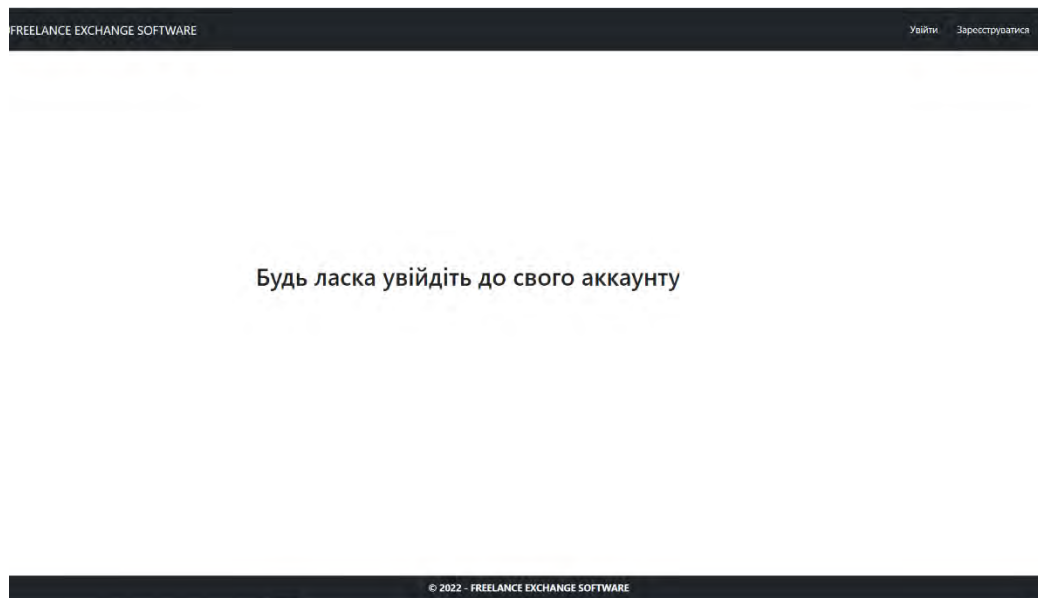


Рисунок 4.1 – Головне вікно, яке відкривається при запуску застосунку

Далі оператор повинен увійти під своїм записом чи зареєструватися, для цього наявні 2 опції у верхньому лівому куті вікна.

Далі розглянемо вікно входу за допомогою електронної адреси та паролю. У цьому вікні присутня як серверна, так і клієнтська валідація. Також у цьому вікні наявна опція для реєстрації нового користувача. Це вікно зображено на рисунку 4.2.

Виконати вхід

Використовуйте локальний обліковий запис для входу.

Адреса електронної пошти

Пароль

☐ Запам'ятати мене

Виконати вхід

[Регістрація нового користувача](#)

© 2022 - FREELANCE EXCHANGE SOFTWARE

Рисунок 4.2 – Вікно, яке дозволяє користувачу авторизуватися у застосунку

Також існує альтернативний потік, коли оператор натискає кнопку зареєструватися. На рисунку 4.3 показано вікно початку реєстрації, де користувач повинен вибрати свою роль шляхом натискання на кнопку.

Реєстрація

Оберіть роль

Замовник

Фрілансер

© 2022 - FREELANCE EXCHANGE SOFTWARE

Рисунок 4.3 – Вікно, яке дозволяє користувачу вибрати одну з наявних ролей на сайті

Далі на буде роглянуто вікно із продовження реєстрації, де користувач вводить основні дані про себе та натискає кнопку зареєструватися. Після цього оператор переадресовується на функціональність, що відповідає кожній з ролей. Також як можна бачити з рисунку, на формі наявна, серверна та клієнтська валідація даних. Далі на рисунку 4.4 показано це вікно.

РЕГІСТРАЦІЯ

Створення нового облікового запису.

- Потрібне поле Ім'я від вашого облікового запису
- Потрібне поле Адреса електронної пошти.
- Потрібен поле Пароль.

Ім'я акаунту

Адрес електронной пошти

Пароль

Підтвердження пароля

Регістрація

© 2022 - FREELANCE EXCHANGE SOFTWARE

Рисунок 4.4 – Вікно, яке дозволяє користувачу вибрати одну з наявних ролей на сайті

Далі розглянемо адміністративну панель. На верхній панелі є посилання на цю сторінку. Посередині вікна наявна область із списком елементарних таблиць, які доступні адміністратору. На рисунку 4.5 зображено функціональність адміністративної панелі адміністратора.

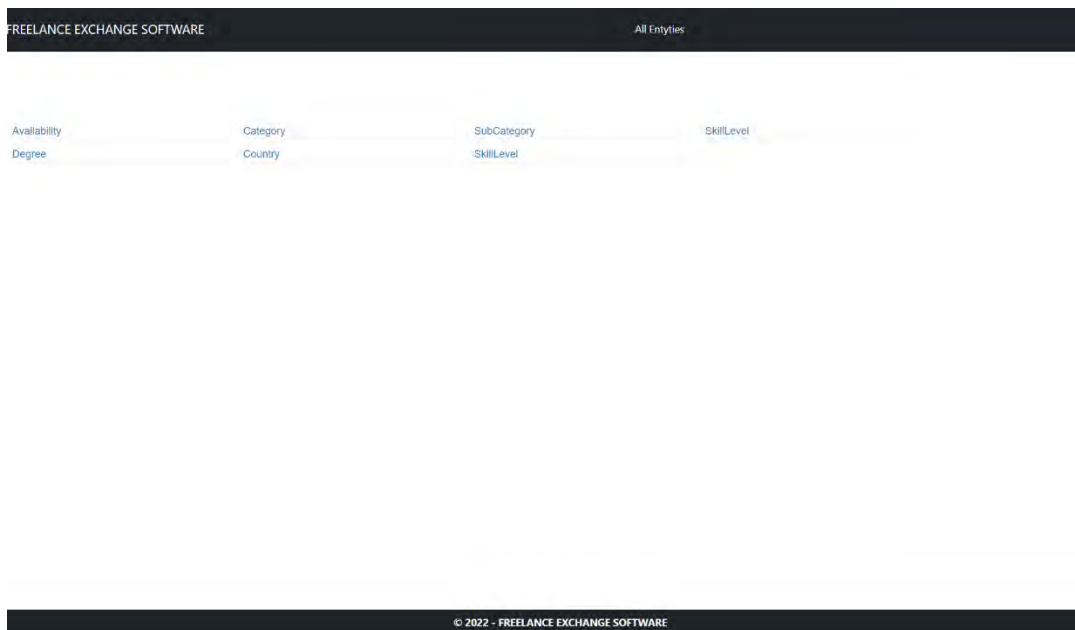


Рисунок 4.5 – Вікно з елементарними таблицями адміністративної панелі

Наступним розглянеться вікно, яке потрібно для виконання операцій з елементарними таблицями. Тут посередині вікна наявна таблиця з усіма наявними записами відносно цієї таблиці, а також поряд елементи керування для додавання, видалення чи редагування запису. На рисунку 4.6 можна побачити вікно для виконання операцій з елементарними таблицями.

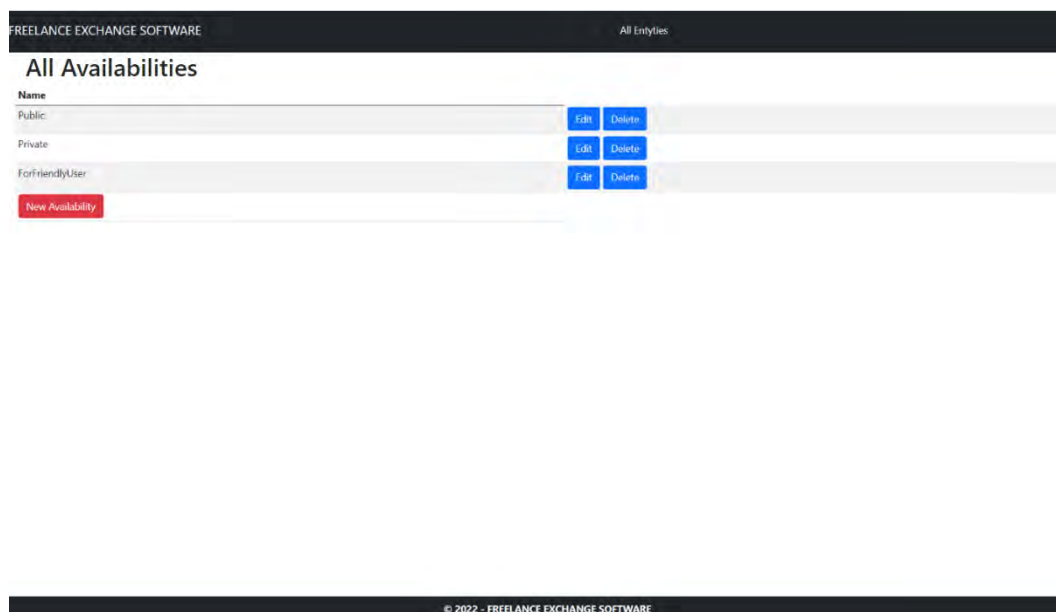


Рисунок 4.6 – Вікно для редагування елементарних записів

Далі розглянемо головне функціональне вікно замовника на якому можна побачити навігаційне меню справа, а також логін у верхньому лівому куті, яке виступає посиланням на зміну профілю. Також там наявна опція вийти. Вікно можна побачити на рисунку 4.7.



Рисунок 4.7 – Головне вікно Замовника

Наступним йде вікно з наявним чатом користувача посередині. Де вони можуть залишати свої коментарі. У верхній частині коментаря пишеться логін користувача. Під коментарями знаходиться форма для заповнення коментаря. Дане вікно можна побачити на рисунку 4.8.

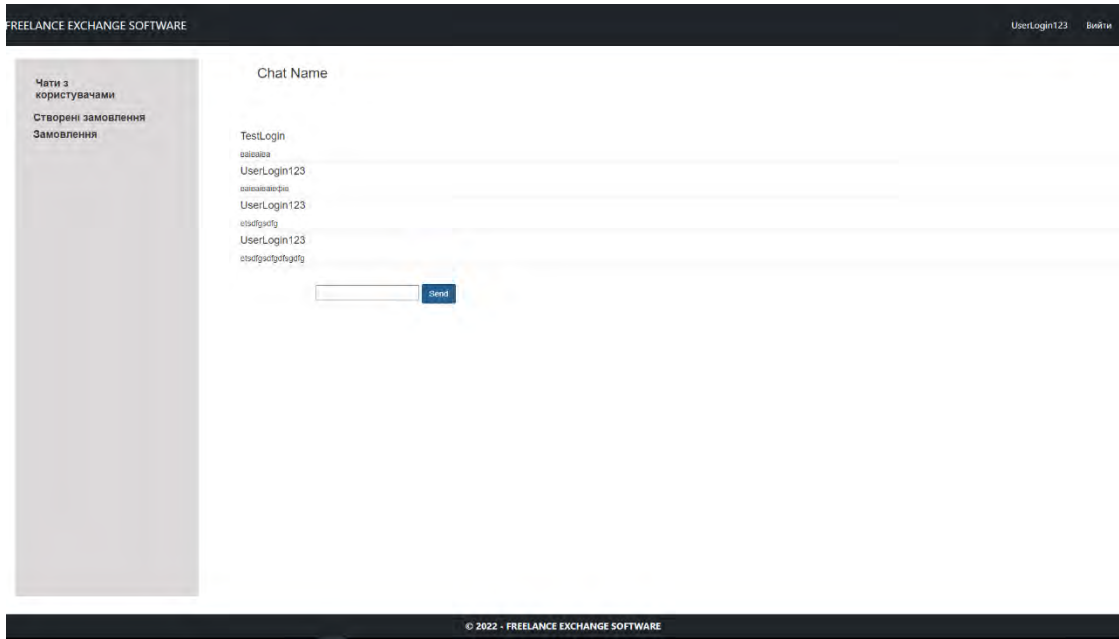


Рисунок 4.8 – Вікно з чатом для замовника

Далі розглянемо вікно з наявною формою для створення замовлення. Ця форма має основні поля для заповнення, які розташовані справа частини вікна. По середині знаходяться група Категорій, у якій можна вибрати категорії замовлення. Далі знаходиться функціональність для доступності цього замовлення іншим користувачам. Вікно можна побачити на рисунку 4.9.

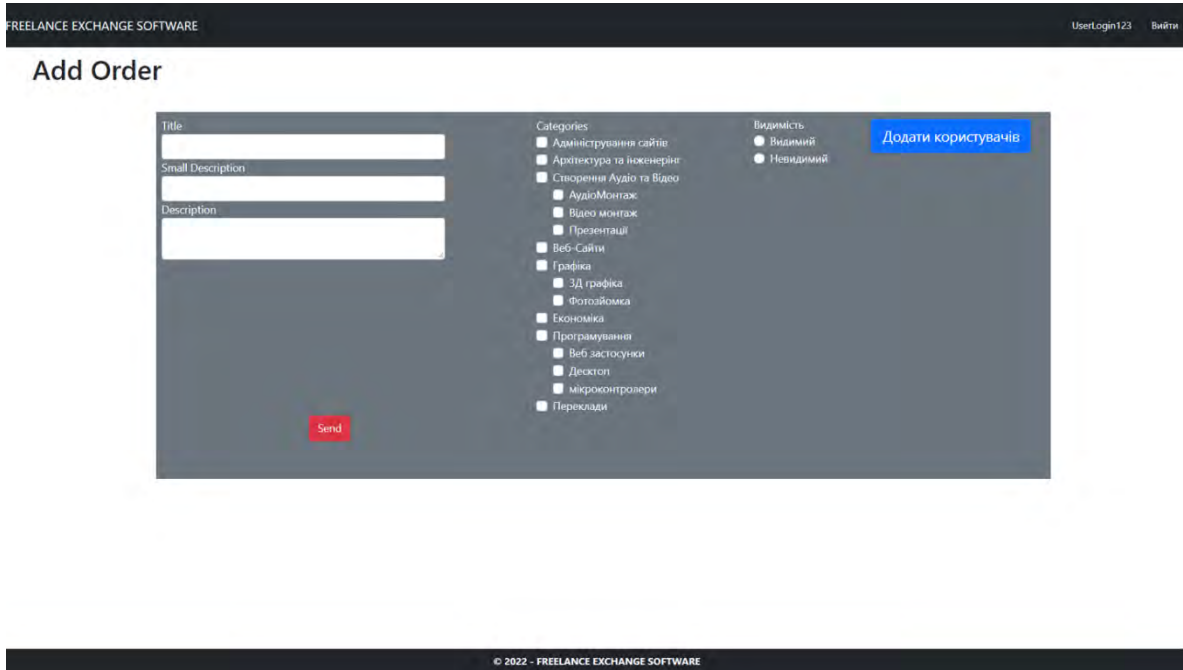


Рисунок 4.9 – Вікно для створення створення замовлення

На рисунку 4.10 зображена форма для роботи із замовленнями, яка на форму до адміністраторської, яка має однакову функціональність.

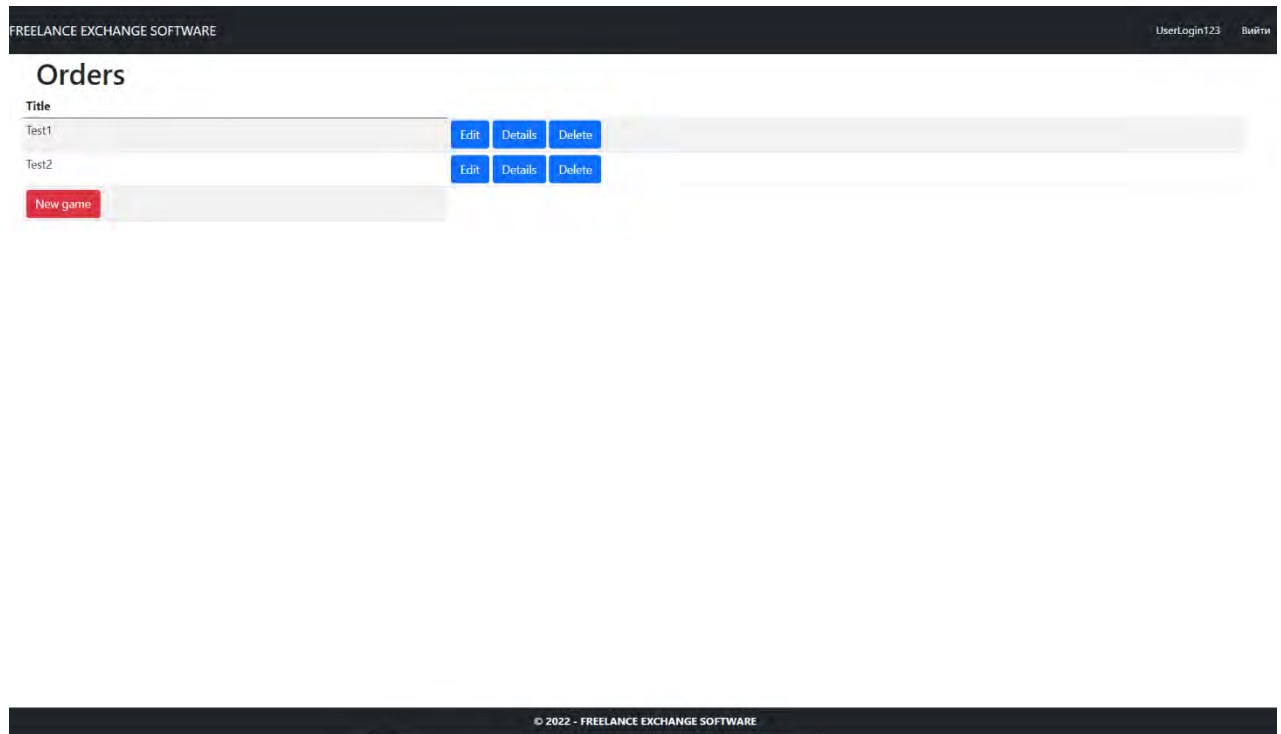


Рисунок 4.10 – Вікно для менеджменту замовлень.

Далі розглянемо форму для додавання фрілансерів до замовлення. Справої сторони розташовані фільтри за допомогою яких можна сортувати фрілансерів. Справа розташовані блоки користувачів з даними логіну рисунком та коли користувач був зареєстрований, а також кнопка додати після якої фрілансер зможе подивитися нове замовлення у себе. Дану форму можна побачити на риснку 4.11.

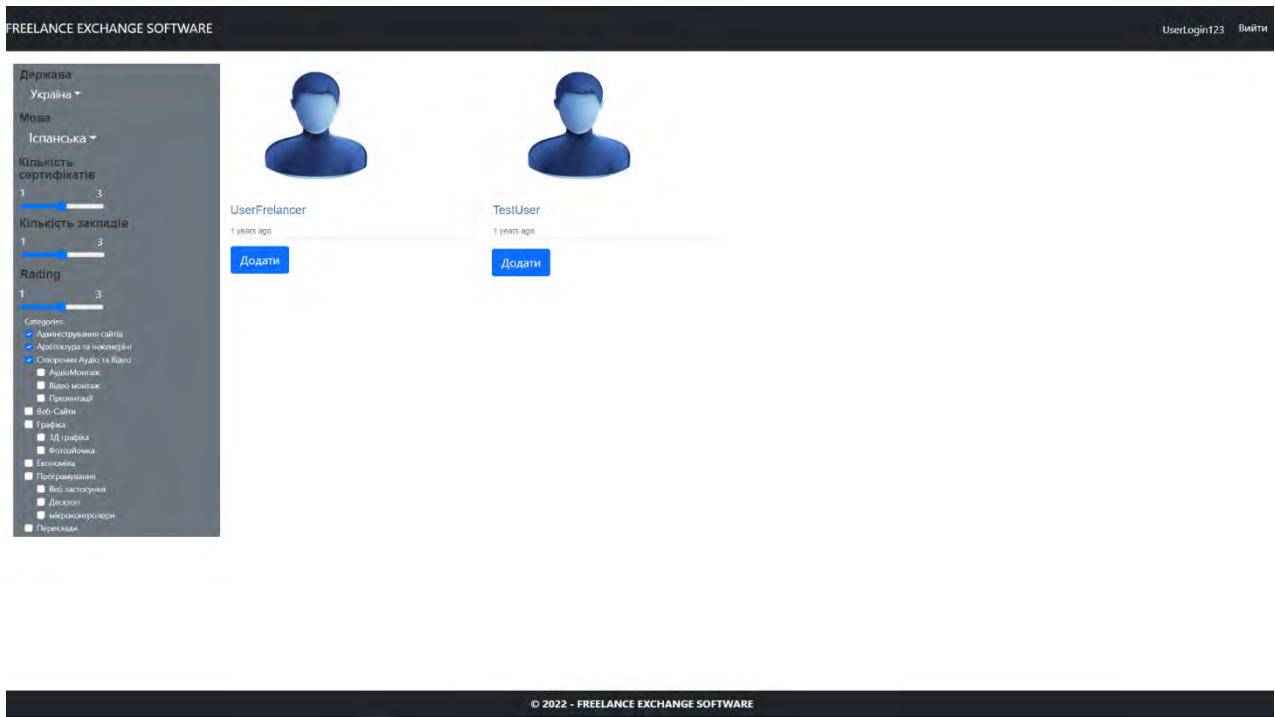


Рисунок 4.11 – Форма для додавання фрілансерів

На рисунку 4.12 зображена форма для редагування замовлення, яка схожа на форму додавання замовлення.

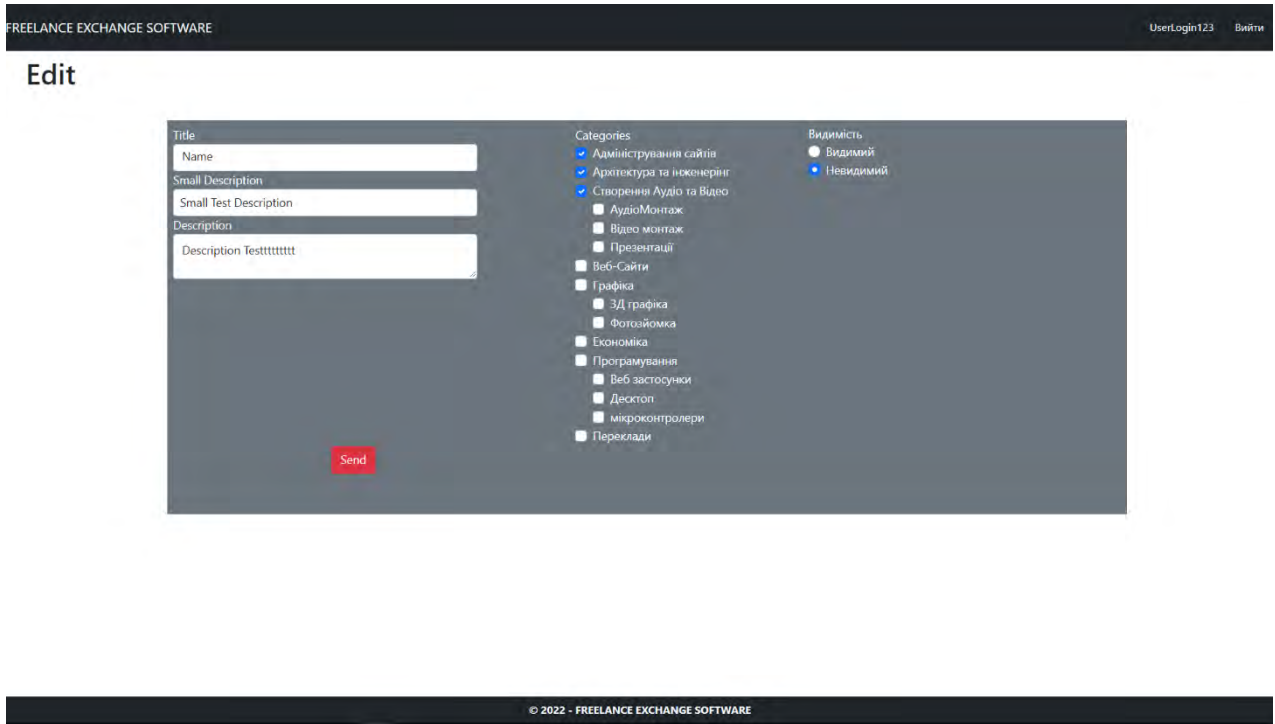


Рисунок 4.12 – Форма для редагування замовлення

На рисунку 4.13 зображена форма для перегляду замовлення та на ній знаходиться можливість для створення нотатки під замовленням.

The screenshot shows a web application interface for 'FREELANCE EXCHANGE SOFTWARE'. The top navigation bar includes the text 'FREELANCE EXCHANGE SOFTWARE' on the left and 'UserLogin123 Вийти' on the right. The main section is titled 'Details'. The form contains the following elements:

- Title:** A text input field with the placeholder 'Name'.
- Small Description:** A text input field with the placeholder 'Small Test Description'.
- Description:** A text input field with the placeholder 'Description Testttttttt'.
- Categories:** A list of categories with checkboxes:
 - ☒ Адміністрування сайтів
 - ☒ Архітектура та інженеринг
 - ☒ Створення Аудіо та Відео
 - ☐ АудіоМонтаж
 - ☐ Відео монтаж
 - ☐ Презентації
 - ☐ Веб-Сайти
 - ☐ Графіка
 - ☐ 3D графіка
 - ☐ Фотозйомка
 - ☐ Економіка
 - ☐ Програмування
 - ☐ Веб застосунки
 - ☐ Десктоп
 - ☐ мікроконтролери
 - ☐ Переклади
- Видимість:** A toggle switch with two options:
 - ☐ Видимий
 - ☒ Невидимий
- Note:** A red button located at the bottom left of the form area.

The footer of the application shows '© 2022 - FREELANCE EXCHANGE SOFTWARE'.

Рисунок 4.13 – Форма для перегляду замовлення

Далі слід розглянути форму для додавання нотаток. У самій формі розташовані нотатки та елементи керування для додавання їх до замовлення. Також варто зазначити, що вікно для запису нататок може розширюватися для потреб користувача. Саме вікно зображено на рисунку 4.14.

The screenshot shows the 'Add Note' form within the Freelance Exchange Software interface. The header bar includes 'FREELANCE EXCHANGE SOFTWARE' on the left and 'UserLogin123 Вийти' on the right. The form title 'Add Note' is displayed in a light blue box. The main form area has a dark gray background and contains a list of notes: 'Тестова нотатка', 'Ще одна нотатка', and 'Нова нотатка'. A red 'Add' button is positioned at the bottom right of the form area. The footer bar shows '© 2022 - FREELANCE EXCHANGE SOFTWARE'.

Рисунок 4.14 – Форма для додавання нотаток

Далі слід розглянути форму для редагування профілю для замовника. На цій формі знаходяться елементи керування для додавання мов та соціальних посилань. Дане вікно зображено на рисунку 4.15.

The screenshot displays the 'Змініть свій профіль' (Edit Profile) form in the Freelance Exchange Software interface. The header bar contains 'FREELANCE EXCHANGE SOFTWARE' and 'UserLogin123 Вийти'. The form title 'Змініть свій профіль' is in a light blue box. The form area has a dark gray background and includes several input fields and buttons. On the left, there is a 'Мови' (Languages) section with a list: 'Англійська', 'Українська', and 'Іспанська', along with a 'Додати' (Add) button. The main section contains fields for 'Ім'я' (Name), 'Прізвище' (Surname), 'По батькові' (Patronymic), 'Дата народження' (Date of birth), 'Держава' (Country) with a dropdown menu showing 'Україна', and 'Місто' (City). On the right, there are fields for social media links: 'https://www.instagram.com/Freelancer', 'https://www.facebook.com/Freelancer', and 'Посилання на соціальну мережу' (Social media link), with a 'Додати' (Add) button. A red 'Змінити' (Save) button is located at the bottom left of the form area. The footer bar shows '© 2022 - FREELANCE EXCHANGE SOFTWARE'.

Рисунок 4.15 – Форма для редагування профілю замовника

Далі слід розглянути форму для редагування профілю фрілансера, яка на відміну від користувачої має ще можливість для додавання скілів, сертифікатів та закладів навчання. Сама форма зображена на рисунку 4.16.

Рисунок 4.16 – Форма для редагування профілю фрілансера

Далі слід розглянути форму для перегляду замовлень фрілансером, яка має у собі список замовлень та керуючі елементи для видалення чи перегляду. Форма перегляду замовлення така ж як і у замовника. Сама форма зображена на рисунку 4.17.

Рисунок 4.17 – Форма для перегляду замовлень, які зав’язані на фрілансера

4.4 Тестування програми

Для тестування програми був використаний підхід TDD. У програмі наявні модульні тести з відповідною структурою до кожного з модулів, тож кількість тестових проєктів відповідає кількості шарів.

У застосунку наявні 3 тестові проєкти:

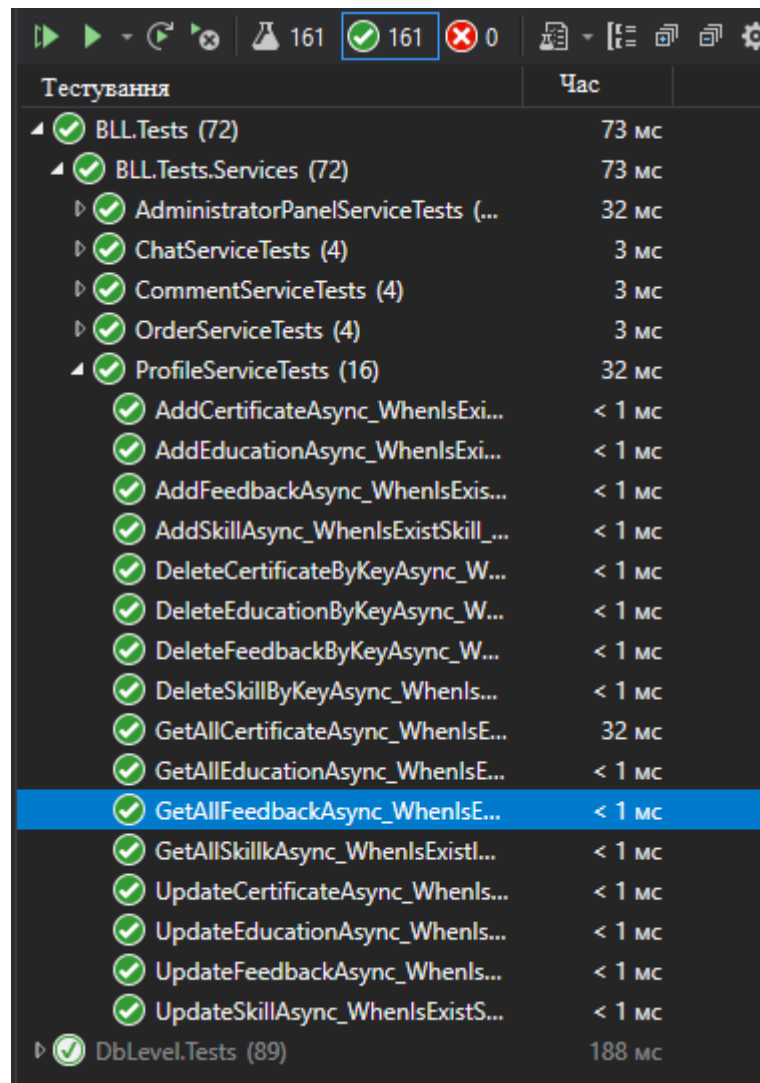
- DbLevel.Tests;
- BLL.Tests;
- UIL.Tests.

Зазвичай тести на цьому рівні досить одноманітні, адже кожен з репозиторіїв наслідується від абстрактного репозиторію, але є можливість, щоб ці методи були перевизначені, тому є сенс проводити такі тестування. На прикладі `AvailabilityRepositoryTests` можна побачити, що тестуються базові функції абстрактного репозитрію і оскільки там немає розгалужень, то одному методу повинен відповідати один тест. На рисунку 4.18 зображений рисунок кількості пройдених модульних тестів для рівня DbLevel.

Тестування	Час
DbLevel.Tests (89)	260 мс
DbLevel.Tests.Repositories.Auxiliary...	162 мс
AvailabilityRepositoryTests (5)	29 мс
AddAsync_NewEntityCreated_A...	29 мс
Delete_WhenEntityDeleted_Ava...	< 1 мс
FindAll_WhenIsExistEntity_IEnu...	< 1 мс
GetByIdAsync_WhenIdsExist_A...	< 1 мс
Update_WhenEntityIsUpdated_...	< 1 мс
CategoryRepositoryTests (5)	< 1 мс
DegreeRepositoryTests (5)	35 мс
ExtensionsTests (4)	< 1 мс
FilePathRepositoryTests (5)	< 1 мс
LanguageRepositoryTests (5)	< 1 мс
NoteRepositoryTests (5)	< 1 мс
SkillLevelRepositoryTests (5)	< 1 мс
SocialLinkRepositoryTests (5)	32 мс
StatusRepositoryTests (5)	29 мс
SubCategoryRepositoryTests (5)	2 мс
CountryRepositoryTests (5)	35 мс
DbLevel.Tests.Repositories.ChatsRe...	98 мс
CertificateRepositoryTests (5)	33 мс
ChatRepositoryTests (5)	< 1 мс
CommentRepositoryTests (5)	63 мс
AddAsync_NewCommentCreat...	< 1 мс
Delete_WhenCommentDeleted...	< 1 мс
FindAll_WhenIsExistEntity_IEnu...	63 мс
GetByIdAsync_WhenIdsExist_A...	< 1 мс
Update_WhenEntityIsUpdated_...	< 1 мс
EducationRepositoryTests (5)	1 мс
FeedbackRepositoryTests (5)	< 1 мс
SkillRepositoryTests (5)	1 мс

Рисунок 4.18 – Зображення пройдених тестів для DAL рівня

В деяких варіантах на цьому рівні може припадати декілька тестів та сценаріїв перевірки. Також методи такого класу можуть генерувати власні виключення, що потім буде оброблено глобальним оброблювачем виключень. На рисунку 4.19 зображено кількість пройдених модульних тестів для рівня BLL.



Тестування	Час
✓ BLL.Tests (72)	73 мс
✓ BLL.Tests.Services (72)	73 мс
✓ AdministratorPanelServiceTests (...)	32 мс
✓ ChatServiceTests (4)	3 мс
✓ CommentServiceTests (4)	3 мс
✓ OrderServiceTests (4)	3 мс
✓ ProfileServiceTests (16)	32 мс
✓ AddCertificateAsync_WhenIsExi...	< 1 мс
✓ AddEducationAsync_WhenIsExi...	< 1 мс
✓ AddFeedbackAsync_WhenIsExis...	< 1 мс
✓ AddSkillAsync_WhenIsExistSkill_...	< 1 мс
✓ DeleteCertificateByKeyAsync_W...	< 1 мс
✓ DeleteEducationByKeyAsync_W...	< 1 мс
✓ DeleteFeedbackByKeyAsync_W...	< 1 мс
✓ DeleteSkillByKeyAsync_WhenIs...	< 1 мс
✓ GetAllCertificateAsync_WhenIsE...	32 мс
✓ GetAllEducationAsync_WhenIsE...	< 1 мс
✓ GetAllFeedbackAsync_WhenIsE...	< 1 мс
✓ GetAllSkillkAsync_WhenIsExistl...	< 1 мс
✓ UpdateCertificateAsync_WhenIs...	< 1 мс
✓ UpdateEducationAsync_WhenIs...	< 1 мс
✓ UpdateFeedbackAsync_WhenIs...	< 1 мс
✓ UpdateSkillAsync_WhenIsExistS...	< 1 мс
✓ DbLevel.Tests (89)	188 мс

Рисунок 4.19 – Зображення пройдених тестів для BLL рівня

Далі можна побачити детальний звіт, який можна масштабувати та який згенерований за допомогою Generator Report. Як можна побачи із звіту, покриття модульними тестами не досить велике – лише 50,3 %, це обумовлено недостатнім покриттям рівня доступу до даних. Також можна побачити дані про кількість розгалужень і дані відносно кількості файлів та класів. Звіт зображений на рисунку 4.20.

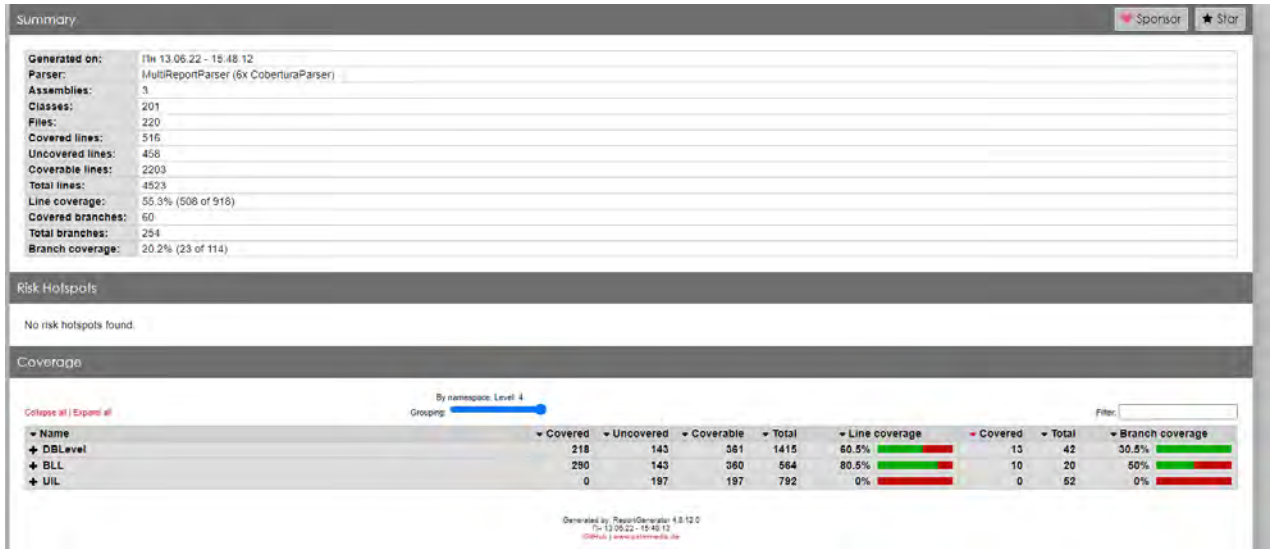


Рисунок 4.20 – Звіт про кількість пройдених тестів та покриття проєкту

ВИСНОВКИ

У результаті роботи було розроблено програмне забезпечення біржі фрілансу. У цьому застосунку учасниками виступають декілька користувачів, такі як фрілансер, адміністратор та замовник.

Був виконаний аналіз предметної області та проблематики цієї сфери. Було встановлено, що наразі ця галузь дуже активно розвивається, проте кількість проблем, з якою стикаються користувачі, доволі велика. Даний застосунок розроблявся для вирішення частини цих проблем. Саме

Для реалізації програми було використано мову програмування C# із застосуванням інтегрованого середовища розробки Visual Studio. Була обрана платформа для вебзастосунку WEB API, що у майбутньому дозволить перейти на більш сучасні технології для інтерфейсної частини, такі як React або Vue та інші. У застосунку використовувалася система керування реляційними БД Microsoft SQL Server та SQLite для тестів. Для адміністрування та звітності використовувалася утиліта SQL Server Management Studio. Для роботи з даними був використаний підхід Code First та фреймворк Entity Framework Core, крім цього були використані бібліотеки AutoMapper, Xunit, AutoFixture, Seriallog, Identity. Для інтерфейсної частини був використаний Bootstrap, мови HTML і CSS, а також фреймворк MVC та генератор сторінок Razor. Був використаний багатoshаровий архітектурний шаблон. Також у застосунку були застосовані принципи SOLID, dependency inversion та dependency injection.

Були представлені базові інтерфейсні вікна для роботи користувача з програмою. Було виконано тестування програми за допомогою модульного тестування, а також загалом визначено покриття тестами.

Програма виконана у вигляді серверного застосунку з використанням клієнт-серверної архітектури, де клієнтом виступає браузер, а сервером розроблюваний застосунок.

Серверний застосунок розроблений для підтримки людей (фрілансерів), у їх прагненні працювати та займатися саморозвитком, також застосунок

прагне надати рівний доступ фрілансерам за допомогою більшої кількості фільтрів, щоб замовник міг з легкістю вибрати саме потрібного фрілансера. Також була додана функція швидкої зміни ролі для користувачів. У майбутньому цей застосунок можна розширити та додати функціональність для вивчення фрілансером популярних платформ саморозвитку, запровадити функціональність для існування великих компаній у цій сфері, а також вирішити правову проблему, яка досить часто зустрічається у даній області.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Биржа фриланса — просто и надежно [Электрон. ресурс]. – Режим доступа : <https://www.weblancer.net>.
2. Шеринговая экономика. Экономика будущего [Электрон. ресурс]. – Режим доступа : <https://vc.ru/finance/319031-sheringovaya-ekonomika-konomika-budushchego>.
3. Sharing economy: A review and agenda for future research [Electronic resource]. – Access mode : <https://www.sciencedirect.com/science/article/abs/pii/S0278431916300743>.
4. Нерегламентована зайнятість населення та способи її регламентування [Електронн. ресурс]. – Режим доступу : <https://works.oklad.ru/view/xJ6ip0aQCjE.html>.
5. Що таке самозайнятість [Електронн. ресурс]. – Режим доступу: <https://fsp.kpi.ua/ua/shho-take-samozajnyatist>.
6. Crowdfunding [Electronic resource]. – Access mode : <https://www.investopedia.com/terms/c/crowdfunding.asp>.
7. Должности C-level [Электрон. ресурс]. – Режим доступа : <https://www.unisender.com/ru/support/about/glossary/dolzhnosti-c-level-kto-takie-smo-ceo-cfo-cvo-cio-i-inye-c>.
8. Outsourcing [Electronic resource]. – Access mode : <https://www.investopedia.com/terms/o/outsourcing.asp>.
9. Кочетов, А.Н. Влияние Интернета на развитие общества [Текст] / А.Н. Кочетов. – Новые стандарты качества жизни, 1999. – С. 43 - 48
10. Шилова Е. В. Обзор фриланс-рынка как составной части шеринговой экономики: сущность, особенности, тенденции развития / Е. В. Шилова, А. И. Боталова // Вестник совета молодых учёных и специалистов Челябинской области. – 2019. – №2 Т. 1. – С. 58–68
11. Statistics on remote workers that will surprise you [Electronic resource]. – Access mode : <https://www.apollotechnical.com/statistics-on-remote-workers>.

12. Essential freelance stats, trends, and facts [Electronic resource]. – Access mode : <https://thrivemyway.com/freelance-stats/#:~:text=Currently%2C%20there%20are%20approximately%201.1,in%20the%20world%20are%20freelancers.>
13. State of freelancing in 2021: statistics you need to know [Electronic resource]. – Access mode : <https://elitecontentmarketer.com/freelance-statistics/>.
14. The self-employed landscape report 2019 [Electronic resource]. – Access mode : <https://www.ipse.co.uk/policy/research/the-self-employed-landscape/the-self-employed-landscape-report-2019.html>.
15. Overview of the european freelance industry [Electronic resource]. – Access mode : https://medium.com/@reshaping_work/overview-of-the-european-freelance-industry-c464b2497960.
16. Review: how to make money on upwork [Electronic resource]. – Access mode : <https://www.nerdwallet.com/article/finance/how-to-make-money-on-upwork#:~:text=What%20is%20Upwork%3F,with%20clients%20and%20get%20paid.>
17. Upwork reviews: real experience from other people [Electronic resource]. – Access mode : <https://productcrafters.io/blog/upwork-reviews/>.
18. Upwork [Electronic resource]. – Access mode : <https://www.upwork.com/>.
19. Friver [Electronic resource]. – Access mode : <https://www.flexjobs.com/guide-freelance-gig-job-platforms/platform/fiverr>.
20. Обзор Fiverr [Электрон. ресурс]. – Режим доступа: <https://ecommerce-platforms.com/ru/articles/fiverr-review>.
21. Friver [Electronic resource]. – Access mode : <https://www.fiverr.com/>.
22. 99Designs review – is it still worth the money [Electronic resource]. – Access mode : <https://www.dreamgrow.com/99designs-review/#:~:text=99designs%20is%20a%20digital%20graphic,design%20services%2C%20such%20as%20printing.>
23. Биржи 99designs.com [Electronic resource]. – Access mode : <https://freelance.today/birgi/99designscom.html>.

24. Обзоры и мнения экспертов 99designs [Электрон. ресурс]. – Режим доступа: <https://www.websiteplanet.com/ru/logo-design-services/99designs/#pricing>.
25. 99Designs [Electronic resource]. – Access mode : <https://99designs.com/>.
26. Company Overview [Electronic resource]. – Access mode : <https://www.freelancer.com/about>.
27. Преимущества фриланс биржи freelancer.com? [Электрон. ресурс]. – Режим доступа: <https://qna.habr.com/q/251064>.
28. Freelancer [Electronic resource]. – Access mode : <https://www.freelancer.com/>.
29. About Peopleperhour [Electronic resource]. – Access mode : <https://www.peopleperhour.com/about>.
30. ОБЗОРЫ И МНЕНИЯ ЭКСПЕРТОВ PeoplePerHour [Электрон. ресурс]. – Режим доступа: <https://www.websiteplanet.com/ru/freelance-websites/peopleperhour/#pricing>.
31. PeoplePerHour [Electronic resource]. – Access mode : <https://www.peopleperhour.com/>.
32. About Guru [Electronic resource]. – Access mode : <https://www.guru.com/about/>.
33. ОБЗОРЫ И МНЕНИЯ ЭКСПЕРТОВ Guru [Электрон. ресурс]. – Режим доступа : <https://www.websiteplanet.com/ru/freelance-websites/guru/#pricing>.
34. Guru [Electronic resource]. – Access mode : <https://www.guru.com/>
35. About ServiceScape [Electronic resource]. – Access mode : <https://www.betterteam.com/servicescape>.
36. ServiceScape [Electronic resource]. – Access mode : <https://www.servicescape.com/>.

ДОДАТОК А
Технічне завдання

Вступ

Сучасне суспільство не стоїть на місці та шукає нові види застосування своїх навичок та вмінь для забезпечення гідного рівня життя: особливо у тяжкий кризовий час чи під час пандемії, коли йде скорочення робочих місць через неспроможність комерційних організацій, а роботодавці застосовують дедалі більші вимоги до працівників при малій заробітній платі, люди починають шукати самореалізацію за іншими формами діяльності.

A.1 Підстави для розробки

Дипломна кваліфікаційна робота виконується за темою «Програмне забезпечення біржі фрілансу» у відповідності з наказом № 102 від 27 квітня 2022 р. за Національним університетом «Запорізька політехніка».

A.2 Призначення розробки

Програма розроблена для підтримки людей (фрілансерів) у їх прагненні працювати та займатися саморозвитком, а також замовників у їх прагненні знайти людину, яка зможе доброчесно виконати замовлення. Для них створені інструменти для створення пропозиції та замовлень.

A.3 Основні вимоги до програмного застосунку

A.3.1 Вимоги до функціональних характеристик

Застосунок має бути розроблений для трьох сутностей:

- адміністратора;
- замовник;
- фрілансер.

Тому можна класифікувати наступні функціональні вимоги у відповідності до вище вказаного списку, а також ввести категорію – спільне або функціональність, яка не прив'язаний до ролі.

Програма повинна задовольняти наступні функції:

- авторизацію та реєстрацію у застосунку;
- швидку зміну ролі шляхом перемикання декількох кнопок;
- мати двунаправлений чат для користувачів та можливість ділитися файлами;
- можливість заповнення мов, якими володіє користувач;
- можливість заповнення держави, дати народження та інших даних при реєстрації.

Програма повинна задовольняти наступні функції для адміністратора:

- заблокувати користувача;
- проводити crud операції з елементарними таблицями;
- блокувати замовлення.

Програма повинна надавати наступні функції для фрілансера:

- можливість виставлення рейтингу після завершення замовлення
- можливість зчитування замовлення;
- проводити crud операції із нотатками до замовлення;
- проводити crud операції для навичок у профілі;
- проводити crud операції для сертифікатів у профілі;
- проводити crud операції для навичок у профілі;
- проводити crud операції для місць навчань у профілі;
- мати можливість для опису себе у профілі;
- переглядати список замовлень по фільтрах;
- можливість залишати та редагувати відгуки;
- можливість видалення та зчитування відгуків.

Програма повинна задовольняти наступні функції для замовника:

- можливість виставлення рейтингу після завершення замовлення;

- проводити студ операції із замовленням;
- проводити студ операції із нотатками до замовлення;
- обмежувати доступ до замовлення;
- проводити студ операції із відгуками у профілі;
- можливість фільтрації та пошуку фрілансерів за здатностями, рейтингом, іменем, мовою та іншим;
- можливість залишати відгук.

A.3.2 Умови експлуатації

Для експлуатації програмного забезпечення потрібно виділити окремий сервер з доступом в інтернет для масового користування. На цьому сервері буде запущений застосунок, який виступає вебсервером і базою даних для постійної роботи з даними користувачів.

A.3.3 Вимоги до складу та параметрів технічних засобів

Мінімальні апаратні вимогами саме до серверної частини:

- оперативна пам'ять: від 8 Гб;
- місце на жорсткому диску під програму: 8 Гб;
- процесор повинен бути Intel чи AMD старше від I5 третього покоління або AMD Ryzen 5 5600H та старше.

A.4. Порядок контролю й приймання

Для контролю виконання дипломної кваліфікаційної роботи повинен використовуватися відповідний календарний план, який затверджено у завданні на роботу. Приймання роботи відбувається на основі проведення процедури захисту роботи.

ДОДАТОК Б
Текст програми

Б.1 Текст файла BaseEntity.cs

```
using System;

namespace DBLevel.Entities
{
    public abstract class BaseEntity
    {
        public Guid Id { get; set; }
    }
}
```

Б.2 Текст файла Availability.cs

```
using DBLevel.Entities;
using System;

namespace DBLevel.EF.Entities.AuxiliaryEntities
{
    public class Availability : BaseEntity
    {
        public string Name { get; set; }

        public Guid? OrderId { get; set; }

        public Order Order { get; set; }
    }
}
```

Б.3 Текст файла Category.cs

```
using DBLevel.Entities;
using System;
using System.Collections.Generic;

namespace DBLevel.EF.Entities.AuxiliaryEntities
{
    public class Category : BaseEntity
    {
        public string Name { get; set; }

        public Guid? OrderId { get; set; }

        public virtual Order Order { get; set; }

        public virtual ICollection<SubCategory> SubCategories { get; set; }
    }
    = new List<SubCategory>();
}
```

Б.4 Текст файла Degree.cs

```
using DBLevel.EF.Entities.ProfileFrelancer;
using DBLevel.Entities;
using System;

namespace DBLevel.EF.Entities.AuxiliaryEntities
{
    public class Degree : BaseEntity
    {

```

```

        public string Name { get; set; }

        public Guid? EducationId { get; set; }

        public Education Education { get; set; }
    }
}

```

Б.5 Текст файла FilePath.cs

```

using DBLevel.EF.Entities.Chats;
using DBLevel.Entities;
using System;

namespace DBLevel.EF.Entities.AuxiliaryEntities
{
    public class FilePath : BaseEntity
    {
        public string FilePathName { get; set; }

        public Guid? NoteId { get; set; }

        public virtual Note Order { get; set; }

        public Guid? CommentId { get; set; }

        public virtual Comment Comment { get; set; }
    }
}

```

Б.6 Текст файла Language.cs

```

using DBLevel.Entities;
using DBLevel.Entities.Abstraction;
using System;

namespace DBLevel.EF.Entities.AuxiliaryEntities
{
    public class Language : BaseEntity
    {
        public string Name { get; set; }

        public Guid? ApplicationUserId { get; set; }

        public ApplicationUser ApplicationUser { get; set; }
    }
}

```

Б.6 Текст файла Note.cs

```

using DBLevel.Entities;
using System;
using System.Collections.Generic;

namespace DBLevel.EF.Entities.AuxiliaryEntities
{
    public class Note : BaseEntity
    {
        public DateTime DataCreation {get;set;}

        public string Title { get; set; }
    }
}

```

```

        public string Description { get; set; }

        public Guid? OrderId { get; set; }

        public virtual Order Order { get; set; }

        public ICollection<FilePath> AttachFiles { get; set; }
    }
}

```

Б.7 Текст файла SkillLevel.cs

```

using DBLevel.EF.Entities.ProfileFrelancer;
using DBLevel.Entities;
using System;

namespace DBLevel.EF.Entities.AuxiliaryEntities
{
    public class SkillLevel : BaseEntity
    {
        public string Name { get; set; }

        public Guid SkillId { get; set; }

        public virtual Skill Skill { get; set; }
    }
}

```

Б.8 Текст файла SocialLink.cs

```

using DBLevel.Entities;
using DBLevel.Entities.Abstraction;
using System;

namespace DBLevel.EF.Entities.AuxiliaryEntities
{
    public class SocialLink : BaseEntity
    {
        public string Name { get; set; }

        public Guid? ApplicationUserId { get; set; }

        public ApplicationUser ApplicationUser { get; set; }
    }
}

```

Б.9 Текст файла Status.cs

```

using DBLevel.Entities;
using System;

namespace DBLevel.EF.Entities.AuxiliaryEntities
{
    public class Status : BaseEntity
    {
        public string Name { get; set; }

        public Guid? OrderId { get; set; }

        public Order Order { get; set; }
    }
}

```

Б.10 Текст файла SubCategory.cs

```
using DBLevel.Entities;
using System;

namespace DBLevel.EF.Entities.AuxiliaryEntities
{
    public class SubCategory : BaseEntity
    {
        public string Name { get; set; }

        public Guid? CategoryId { get; set; }

        public virtual Category Category { get; set; }
    }
}
```

Б.11 Текст файла Country.cs

```
using DBLevel.EF.Entities.ProfileFrelancer;
using DBLevel.Entities;
using System;

namespace DBLevel.EF.Entities.AuxiliaryEntities
{
    public class Country : BaseEntity
    {
        public string Name { get; set; }

        public Guid? EducationId { get; set; }

        public Education Education { get; set; }
    }
}
```

Б.12 Текст файла Chat.cs

```
using DBLevel.EF.Entities.LinkTables;
using DBLevel.Entities;
using DBLevel.Entities.Abstraction;
using System;
using System.Collections.Generic;

namespace DBLevel.EF.Entities.Chats
{
    public class Chat : BaseEntity
    {
        public string Title { get; set; }

        public DateTime DateCreation { get; set; }

        public virtual ICollection<Comment> Commemnts { get; set; } = new
List<Comment>();

        public virtual ICollection<ChatAppUser> ApplicationUsers { get;
set; } = new List<ChatAppUser>();
    }
}
```

Б.13 Текст файла Comment.cs

```
namespace DBLevel.EF.Entities.Chats
{
    public class Comment : BaseEntity
    {
        public DateTime DateCration { get; set; }

        public string Description { get; set; }

        public Guid? ChatId { get; set; }

        public Guid? ApplicationUserId { get; set; }

        public virtual ApplicationUser ApplicationUser { get; set; }

        public Chat Chat { get; set; }

        public ICollection<FilePath> AttachFiles { get; set; }
    }
}
```

Б.14 Текст файла ApplicationUser.cs

```
using DBLevel.EF.Entities.AuxiliaryEntities;
using DBLevel.EF.Entities.Chats;
using DBLevel.EF.Entities.LinkTables;
using DBLevel.Entities.Users;
using Microsoft.AspNetCore.Identity;
using System;
using System.Collections.Generic;

namespace DBLevel.Entities.Abstraction
{
    public class ApplicationUser : IdentityUser<Guid>
    {
        public string Name { get; set; }

        public string Surname { get; set; }

        public string PicturePath { get; set; }

        public string Patronymic { get; set; }

        public DateTime BirthYear { get; set; }

        public string Country { get; set; }

        public string City { get; set; }

        public DateTime DateRegistration { get; set; }

        public Guid? FreelancerId { get; set; }

        public Guid? CustomerId { get; set; }

        public Guid? AdministratorId { get; set; }

        public Freelancer Freelancer { get; set; }
    }
}
```

```

        public Customer Customer { get; set; }

        public Administrator Administrator { get; set; }

        public virtual ICollection<ApplicationRole> Roles { get; } = new
List<ApplicationRole>();

        public virtual ICollection<Language> Languages { get; set; } = new
List<Language>();

        public virtual ICollection<SocialLink> SocialLinks { get; set; } =
new List<SocialLink>();

        public virtual ICollection<ChatAppUser> Chats { get; set; } = new
List<ChatAppUser>();

        public virtual ICollection<Comment> Comments { get; set; } = new
List<Comment>();
    }
}

```

Б.15 Текст файла Administrator.cs

```

using DBLevel.Entities.Abstraction;
using System;

namespace DBLevel.Entities
{
    public class Administrator : BaseEntity
    {
        public Guid? ApplicationUserId { get; set; }

        public virtual ApplicationUser ApplicationUser { get; set; }
    }
}

```

Б.16 Текст файла ApplicationRole.cs

```

using DBLevel.Entities.Abstraction;
using Microsoft.AspNetCore.Identity;
using System;

namespace DBLevel.Entities.Users
{
    public class ApplicationRole : IdentityRole<Guid>
    {
        public Guid? ApplicationUserId { get; set; }

        public ApplicationUser ApplicationUser { get; set; }
    }
}

```

Б.17 Текст файла Freelancer.cs

```

using DBLevel.EF.Entities.LinkTables;
using DBLevel.EF.Entities.ProfileFrelander;
using DBLevel.Entities.Abstraction;
using System;
using System.Collections.Generic;

namespace DBLevel.Entities

```

```

{
    public class Freelancer : BaseEntity
    {
        public int Raiting { get; set; }

        public string ProfileDescription { get; set; }

        public Guid? ApplicationUserId { get; set; }

        public virtual ApplicationUser ApplicationUser { get; set; }

        public virtual ICollection<Skill> Skills { get; set; } = new
List<Skill>();

        public virtual ICollection<Education> Educations { get; set; } =
new List<Education>();

        public virtual ICollection<Certificate> Certificates { get; set; }
= new List<Certificate>();

        public virtual ICollection<FreelancerOrder> Orders { get; set; } =
new List<FreelancerOrder>();

        public virtual ICollection<Feedback> Feedbacks { get; set; } = new
List<Feedback>();
    }
}

```

Б.17 Текст файла ChatAppUser.cs

```

using DBLevel.EF.Entities.Chats;
using DBLevel.Entities;
using DBLevel.Entities.Abstraction;
using System;

namespace DBLevel.EF.Entities.LinkTables
{
    public class ChatAppUser : BaseEntity
    {
        public Guid ChatId { get; set; }

        public virtual Chat Chat { get; set; }

        public Guid ApplicationUserId { get; set; }

        public virtual ApplicationUser ApplicationUser { get; set; }
    }
}

```

Б.17 Текст файла ChatAppUser.cs

```

using DBLevel.Entities;
using System;

namespace DBLevel.EF.Entities.LinkTables
{
    public class FreelancerOrder : BaseEntity
    {
        public Guid OrderId { get; set; }

        public virtual Order Order { get; set; }

        public Guid FreelancerId { get; set; }
    }
}

```

```

        public virtual Freelancer Freelancer { get; set; }
    }
}

```

Б.18 Текст файла Certificate.cs

```

using DBLevel.Entities;
using System;

namespace DBLevel.EF.Entities.ProfileFrelancer
{
    public class Certificate : BaseEntity
    {
        public string Descriptions { get; set; }

        public string PathToImg { get; set; }

        public Guid? FreelancerId { get; set; }

        public virtual Freelancer Freelancer { get; set; }
    }
}

```

Б.19 Текст файла Education.cs

```

using DBLevel.EF.Entities.AuxiliaryEntities;
using DBLevel.Entities;
using System;

namespace DBLevel.EF.Entities.ProfileFrelancer
{
    public class Education : BaseEntity
    {
        public string NameUniversity { get; set; }

        public Guid? CountryId { get; set; }

        public Guid? DegreeId { get; set; }

        public Guid? FreelancerId { get; set; }

        public virtual Degree Degree { get; set; }

        public virtual Country Country { get; set; }

        public virtual Freelancer Freelancer { get; set; }
    }
}

```

Б.20 Текст файла Feedback.cs

```

using DBLevel.Entities;
using DBLevel.Entities.Abstraction;
using System;

namespace DBLevel.EF.Entities.ProfileFrelancer
{
    public class Feedback : BaseEntity
    {
        public string Description { get; set; }
    }
}

```

```

        public DateTime Date { get; set; }

        public Guid? CustomerUserId { get; set; }

        public Guid? FreelancerId { get; set; }

        public virtual Freelancer Freelancer { get; set; }

        public virtual Customer Customer { get; set; }
    }
}

```

Б.21 Текст файла Skill.cs

```

using DBLevel.EF.Entities.AuxiliaryEntities;
using DBLevel.Entities;
using System;

namespace DBLevel.EF.Entities.ProfileFrelancer
{
    public class Skill : BaseEntity
    {
        public string Description { get; set; }

        public Guid? SkillLevelId { get; set; }

        public Guid? FreelancerId { get; set; }

        public virtual SkillLevel SkillLevel { get; set; }

        public virtual Freelancer Freelancer { get; set; }
    }
}

```

Б.21 Текст файла Order.cs

```

using DBLevel.EF.Entities.AuxiliaryEntities;
using DBLevel.EF.Entities.LinkTables;
using System;
using System.Collections.Generic;

namespace DBLevel.Entities
{
    public class Order : BaseEntity
    {
        public DateTime DateCration { get; set; }

        public string Title { get; set; }

        public string SmallDescription {get; set;}

        public string Description { get; set; }

        public Guid? AvailabilityId { get; set; }

        public Guid? StatusId { get; set; }

        public Guid? CustomerId { get; set; }

        public virtual Availability Availability { get; set; }

        public virtual Status Status { get; set; }
    }
}

```

```

        public virtual Customer Customer { get; set; }

        public virtual ICollection<Category> Category { get; set; } = new
List<Category>();

        public virtual ICollection<Note> Notes { get; set; } = new
List<Note>();

        public virtual ICollection<FreelancerOrder> Freelancers { get; set;
} = new List<FreelancerOrder>();
    }
}

```

Б.22 Текст файла FESContext.cs

```

using DBLevel.EF.Entities.AuxiliaryEntities;
using DBLevel.EF.Entities.Chats;
using DBLevel.EF.Entities.LinkTables;
using DBLevel.EF.Entities.ProfileFreelancer;
using DBLevel.Entities;
using DBLevel.Entities.Abstraction;
using DBLevel.Entities.Users;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using System;

namespace DBLevel.EF
{
    public class FESContext : IdentityDbContext<ApplicationUser,
ApplicationRole, Guid>
    {
        public FESContext(DbContextOptions<FESContext> options)
            : base(options)
        {
            Database.EnsureCreated();
        }

        public DbSet<Order> Orders { get; set; }

        public DbSet<ChatAppUser> ChatAppUsers { get; set; }

        public DbSet<FreelancerOrder> FreelancerOrders { get; set; }

        #region AuxiliaryEntities

        public DbSet<Category> Categories { get; set; }

        public DbSet<Degree> Degrees { get; set; }

        public DbSet<Degree> Language { get; set; }

        public DbSet<Note> Note { get; set; }

        public DbSet<SkillLevel> SkillLevel { get; set; }

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

        public DbSet<SubCategory> SubCategory { get; set; }

        public DbSet<Country> Country { get; set; }

        #endregion
    }
}

```

```

        #region Chats

        public DbSet<Chat> Chat { get; set; }

        public DbSet<Comment> Comments { get; set; }

        #endregion

        #region ProfileFrelancer

        public DbSet<Certificate> Certificate { get; set; }

        public DbSet<Education> Education { get; set; }

        public DbSet<Feedback> Feedback { get; set; }

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

        #endregion

        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);

            builder.ApplyConfigurationsFromAssembly(typeof(FESContext).Assembly);
        }
    }
}

```

Б.22 Текст файла DateTimeProvider.cs

```

using System;

namespace DBLevel.Extensions
{
    public static class DateTimeProvider
    {
        public static DateTime GenerateDateTimeNow()
        {
            return DateTime.UtcNow;
        }
    }
}

```

Б.23 Текст файла IdGenerator.cs

```

using System;

namespace DBLevel.Extensions
{
    public static class IdGenerator
    {
        public static Guid GenereteId(string str = "")
        {
            Guid id;

            if (str == string.Empty)
            {
                return Guid.NewGuid();
            }

            if (!Guid.TryParse(str, out id))
                id = new Guid(str);
        }
    }
}

```

```

        return id;
    }
}

```

Б.23 Текст файла IUnitOfWork.cs

```

using DBLevel.EF;
using DBLevel.Repositories.AuxiliaryRepositories.Interfaces;
using DBLevel.Repositories.ChatsRepositories.Interfaces;
using DBLevel.Repositories.ProfileFrelancerRepositories.Interfaces;
using System.Threading.Tasks;

/// <summary>
/// Interfaces IGenericRepository and IUnitOfWork
/// </summary>
namespace DBLevel
{
    /// <summary>
    /// public generic interface IUnitOfWork
    /// implements fields methods for managing repositories
    /// </summary>
    public interface IUnitOfWork
    {
        FESContext IdentityDbContext { get; set; }

        public IAvailabilityRepository AvailabilityRepository { get; }

        public ICategoryRepository CategoryRepository { get; }

        public IDegreeRepository DegreeRepository { get; }

        public IFilePathRepository FilePathRepository { get; }

        public INoteRepository NoteRepository { get; }

        public ISkillLevelRepository SkillLevelRepository { get; }

        public ISocialLinkRepository SocialLinkRepository { get; }

        public IStatusRepository StatusRepository { get; }

        public ISubCategoryRepository SubCategoryRepository { get; }

        public ICountryRepository CountryRepository { get; }

        public IChatRepository ChatRepository { get; }

        public ICommentRepository CommentRepository { get; }

        public ICertificateRepository CertificateRepository { get; }

        public IEducationRepository EducationRepository { get; }

        public ISkillRepository SkillRepository { get; }

        public IFeedbackRepository FeedbackRepository { get; }

        /// <summary>
        /// Asynchronously saves all changes made in this context to the
        underlying database.
        /// </summary>
    }
}

```

```

        /// <returns> The task result contains the number of state entries
        written to the underlying database.</returns>
        Task<int> SaveChangesAsync();
    }
}

```

Б.23 Текст файла IUnitOfWork.cs

```

using DBLevel.EF;
using DBLevel.Repositories.AuxiliaryEntities;
using DBLevel.Repositories.AuxiliaryRepositories;
using DBLevel.Repositories.AuxiliaryRepositories.Interfaces;
using DBLevel.Repositories.ChatsRepositories;
using DBLevel.Repositories.ChatsRepositories.Interfaces;
using DBLevel.Repositories.ProfileFrelancerRepositories;
using DBLevel.Repositories.ProfileFrelancerRepositories.Interfaces;
using System;
using System.Threading.Tasks;

/// <summary>
/// name space that contains the class UnitOfWork
/// </summary>
namespace DBLevel
{
    /// <summary>
    /// genetic class UnitOfWork which is needed to manage repositories
    /// inheritance interface IUnitOfWork
    /// </summary>
    public class UnitOfWork : IUnitOfWork
    {
        public FESContext IdentityDbContext { get; set; }

        /// <summary>
        /// constructor UnitOfWork that takes Context of type FESContext
        and bind DbContext
        /// </summary>
        /// <param name="FESContext"></param>
        public UnitOfWork(FESContext IdentityDbContext)
        {
            this.IdentityDbContext = IdentityDbContext;
        }

        #region Auxiliary

        private IAvailabilityRepository availabilities;

        private ICategoryRepository categories;

        private IDegreeRepository deegres;

        private IFilePathRepository filePath;

        private INoteRepository notes;

        private ISkillLevelRepository skillLevels;

        private ISocialLinkRepository socialLinks;

        private IStatusRepository statuses;

        private ISubCategoryRepository subCategories;

        private ICountryRepository countries;

```

```

        public IAvailabilityRepository AvailabilityRepository =>
availabilities ??= new AvailabilityRepository(IdentityDbContext);

        public ICategoryRepository CategoryRepository => categories ??= new
CategoryRepository(IdentityDbContext);

        public IDegreeRepository DegreeRepository => deegres ??= new
DegreeRepository(IdentityDbContext);

        public IFilePathRepository FilePathRepository => filesPath ??= new
FilePathRepository(IdentityDbContext);

        public INoteRepository NoteRepository => notes ??= new
NoteRepository(IdentityDbContext);

        public ISkillLevelRepository SkillLevelRepository => skillLevels
??= new SkillLevelRepository(IdentityDbContext);

        public ISocialLinkRepository SocialLinkRepository => socialLinks
??= new SocialLinkRepository(IdentityDbContext);

        public IStatusRepository StatusRepository => statuses ??= new
StatusRepository(IdentityDbContext);

        public ISubCategoryRepository SubCategoryRepository =>
subCategories ??= new SubCategoryRepository(IdentityDbContext);

        public ICountryRepository CountryRepository => countries ??= new
CountryRepository(IdentityDbContext);

        #endregion

        #region Chats

        private IChatRepository chats;

        private ICommentRepository comments;

        public IChatRepository ChatRepository => chats ??= new
ChatRepository(IdentityDbContext);

        public ICommentRepository CommentRepository => comments ??= new
CommentRepository(IdentityDbContext);

        #endregion

        #region ProfileFrelancer

        private ICertificateRepository certificates;

        private IEducationRepository educations;

        private IFeedbackRepository feedbacks;

        private ISkillRepository skills;

        public ICertificateRepository CertificateRepository => certificates
??= new CertificateRepository(IdentityDbContext);

        public IEducationRepository EducationRepository => educations ??=
new EducationRepository(IdentityDbContext);

```

```

        public ISkillRepository SkillRepository => skills ??= new
SkillRepository(IdentityDbContext);

        public IFeedbackRepository FeedbackRepository => feedbacks ??= new
FeedbackRepository(IdentityDbContext);

    #endregion

    /// <summary>
    /// Save asynchronous all changes in DB
    /// </summary>
    /// <returns>Return Data with amount of change entities </returns>
    public async Task<int> SaveChangesAsync()
    {
        return await IdentityDbContext.SaveChangesAsync();
    }

    private bool _disposed = false;

    /// <summary>
    /// method for clear object
    /// </summary>
    public void Dispose()
    {
        Dispose(true);

        GC.SuppressFinalize(this);
    }

    /// <summary>
    /// protected virtual to override logic IDispose
    /// </summary>
    /// <param name="disposing"></param>
    protected virtual void Dispose(bool disposing)
    {
        if (!_disposed && disposing)
        {
            IdentityDbContext.Dispose();
        }

        _disposed = true;
    }
}
}

```

Б.24 Текст файла IGenericRepository.cs

```

using DBLevel.Entities;
using System;
using System.Linq;
using System.Threading.Tasks;

/// <summary>
/// Name space Abstraction Interfaces
/// </summary>
namespace DBLevel.Repositories.Abstraction.Interfaces
{
    /// <summary>
    /// Interface IGenericRepository
    /// Crud operation which general for each repository
    /// </summary>
    /// <typeparam name="TEntity"></typeparam>
    public interface IGenericRepository<TEntity> where TEntity : BaseEntity

```

```

{
    /// <summary>
    /// signature method
    /// </summary>
    /// <returns>collection TEntity</returns>
    IQueryable<TEntity> FindAll();

    /// <summary>
    /// signature method
    /// </summary>
    /// <param name="entity"></param>
    /// <returns> Task </returns>
    Task AddAsync(TEntity entity);

    /// <summary>
    /// signature method
    /// </summary>
    /// <param name="entity"></param>
    void Update(TEntity entity);

    /// <summary>
    /// signature method
    /// </summary>
    /// <param name="entity"></param>
    void Delete(TEntity entity);

    /// <summary>
    /// signature method
    /// </summary>
    /// <param name="id"></param>
    /// <returns></returns>
    Task<TEntity> GetByIdAsync(Guid id);

    /// <summary>
    /// signature method
    /// </summary>
    /// <param name="id"></param>
    /// <returns></returns>
    Task DeleteByIdAsync(Guid id);
}
}

```

Б.25 Текст файла GenericRepository.cs

```

using DBLevel.EF;
using DBLevel.Entities;
using DBLevel.Repositories.Abstraction.Interfaces;
using Microsoft.EntityFrameworkCore;
using System;
using System.Linq;
using System.Threading.Tasks;

namespace DBLevel.Repositories.Abstraction
{
    public class GenericRepository<TEntity> : IDisposable,
    IGenericRepository<TEntity>
        where TEntity : BaseEntity
    {
        private readonly FESContext _context;
        private bool _disposed;

        public GenericRepository(FESContext context)
        {

```

```

        _context = context;
    }

    public virtual IQueryable<TEntity> FindAll()
    {
        return _context.Set<TEntity>();
    }

    public virtual async Task AddAsync(TEntity entity)
    {
        await _context.Set<TEntity>().AddAsync(entity);
    }

    public virtual void Delete(TEntity entity)
    {
        _context.Set<TEntity>().Remove(entity);
    }

    public virtual void Update(TEntity entity)
    {
        _context.Entry(entity).State = EntityState.Modified;
    }

    public virtual async Task<TEntity> GetByIdAsync(Guid id)
    {
        return await _context.Set<TEntity>().FindAsync(id);
    }

    public virtual async Task DeleteByIdAsync(Guid id)
    {
        var entity = await GetByIdAsync(id);
        _context.Set<TEntity>().Remove(entity);
    }

    public void Dispose() => Dispose(true);

    protected virtual void Dispose(bool disposing)
    {
        if (!_disposed)
        {
            if (disposing)
            {
                _context.Dispose();
            }

            _disposed = true;
        }
    }
}

```

Б.26 Текст файла IGenericRepository.cs

```

using DBLevel.Entities;
using System;
using System.Linq;
using System.Threading.Tasks;

/// <summary>
/// Name space Abstraction Interfaces
/// </summary>
namespace DBLevel.Repositories.Abstraction.Interfaces
{

```

```

    /// <summary>
    /// Interface IGenericRepository
    /// Crud operation which general for each repository
    /// </summary>
    /// <typeparam name="TEntity"></typeparam>
    public interface IGenericRepository<TEntity> where TEntity : BaseEntity
    {
        /// <summary>
        /// signature method
        /// </summary>
        /// <returns>collection TEntity</returns>
        IQueryable<TEntity> FindAll();

        /// <summary>
        /// signature method
        /// </summary>
        /// <param name="entity"></param>
        /// <returns> Task </returns>
        Task AddAsync(TEntity entity);

        /// <summary>
        /// signature method
        /// </summary>
        /// <param name="entity"></param>
        void Update(TEntity entity);

        /// <summary>
        /// signature method
        /// </summary>
        /// <param name="entity"></param>
        void Delete(TEntity entity);

        /// <summary>
        /// signature method
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        Task<TEntity> GetByIdAsync(Guid id);

        /// <summary>
        /// signature method
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        Task DeleteByIdAsync(Guid id);
    }
}

```

Б.27 Текст файла CommentRepository.cs

```

using DBLevel.EF;
using DBLevel.EF.Entities.Chats;
using DBLevel.Repositories.Abstraction;
using DBLevel.Repositories.ChatsRepositories.Interfaces;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;

namespace DBLevel.Repositories.ChatsRepositories
{
    public class CommentRepository : GenericRepository<Comment>,
        ICommentRepository

```

```

    {
        public CommentRepository(FESContext fESContext) : base(fESContext)
    { }

        public IEnumerable<Comment> FindAllByChatIdWithDetails(Guid id)
        {
            return FindAll().Where(el => el.ChatId == id)
                .Include(r => r.ApplicationUser)
                .Include(r => r.Chat);
        }
    }
}

```

Б.28 Текст файла ChatRepository.cs

```

using DBLevel.EF;
using DBLevel.EF.Entities.Chats;
using DBLevel.Repositories.Abstraction;
using DBLevel.Repositories.ChatsRepositories.Interfaces;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;

namespace DBLevel.Repositories.ChatsRepositories
{
    public class ChatRepository : GenericRepository<Chat>, IChatRepository
    {
        public ChatRepository(FESContext fESContext) : base(fESContext) { }

        public IEnumerable<Chat> FindAllWithDetails(Expression<Func<Chat,
bool>> expression)
        {
            return FindAll().Where(expression)
                .Include(r => r.ApplicationUsers)
                .Include(r => r.Commemnts);
        }
    }
}

```

Б.29 Текст файла ChatRepository.cs

```

using DBLevel.EF;
using DBLevel.EF.Entities.Chats;
using DBLevel.Repositories.Abstraction;
using DBLevel.Repositories.ChatsRepositories.Interfaces;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;

namespace DBLevel.Repositories.ChatsRepositories
{
    public class ChatRepository : GenericRepository<Chat>, IChatRepository
    {
        public ChatRepository(FESContext fESContext) : base(fESContext) { }

        public IEnumerable<Chat> FindAllWithDetails(Expression<Func<Chat,
bool>> expression)
        {
            return FindAll().Where(expression)

```

```

        .Include(r => r.ApplicationUsers)
        .Include(r => r.Commemnts);
    }
}

```

Б.30 Текст файла InputCommentDTO.cs

```

using System;
using System.Collections.Generic;

namespace BLL.DTO.In
{
    public class InputCommentDTO
    {
        public Guid Id { get; set; }

        public string Description { get; set; }

        public Guid ApplicationUserId { get; set; }

        public Guid ChatId { get; set; }

        public IEnumerable<string> AttachFiles { get; set; }
    }
}

```

Б.30 Текст файла InputCommentDTO.cs

```

using System;
using System.Collections.Generic;

namespace BLL.DTO.In
{
    public class InputCommentDTO
    {
        public Guid Id { get; set; }

        public string Description { get; set; }

        public Guid ApplicationUserId { get; set; }

        public Guid ChatId { get; set; }

        public IEnumerable<string> AttachFiles { get; set; }
    }
}

```

Б.31 Текст файла Guard.cs

```

using System;

namespace BLL.Exeption
{
    public static class Guard
    {
        public static void VerifyNullObject(object verifyObj, string
method, string parentClass)
        {
            if (verifyObj == null)
            {

```

```

        throw new ArgumentNullException(paramName:
nameof(verifyObj), message: $"Object is null. Class = {parentClass}, method
= {method}");
    }
}

    public static void VerifyNullOrEmpthyString(string verifyStr,
string method, string parentClass)
    {
        if (String.IsNullOrEmpty(verifyStr))
        {
            throw new ArgumentNullException(paramName:
nameof(verifyStr), message: $"key is null or empty. Class = {parentClass},
method = {method}");
        }
    }

    public static void VerifyEmpthyId(Guid id, string method, string
parentClass)
    {
        if (id == Guid.Empty)
        {
            throw new ArgumentNullException(paramName: nameof(id),
message: $"id is empty. Class = {parentClass}, method = {method}");
        }
    }
}

```

Б.31 Текст файла IChatService.cs

```

using BLL.DTO.Input.Chat;
using BLL.DTO.Output.Chat;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace BLL.Services.Interfaces.Chat
{
    public interface IChatService
    {
        Task AddChatWithMembersAsync(string title,
List<OutputApplicationUserDTO> applicationUsers);

        Task DeleteChatAsync(Guid id);

        Task UpdateChatTitleAsync(Guid id, string title);

        Task AddMembersToChatAsync(Guid id, List<OutputApplicationUserDTO>
applicationUsers);

        Task<InnputChatDTO> GetAllChatsByUserId(Guid id);
    }
}

```

Б.32 Текст файла ICommentService.cs

```

using BLL.DTO.Chat.Out;
using BLL.DTO.In;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

```

```

namespace BLL.Services.Chat.Interfaces
{
    public interface ICommentService
    {
        Task AddAsync(InputCommentDTO model);

        Task UpdateAsync(InputCommentDTO model);

        Task DeleteByKeyAsync(Guid id);

        Task<IEnumerable<OutputCommentDTO>> GetAllByChatIdAsync(Guid id);
    }
}

```

Б.33 Текст файла ChatService.cs

```

using AutoMapper;
using BLL.DTO.Input.Chat;
using BLL.DTO.Output.Chat;
using BLL.Services.Interfaces.Chat;
using DBLevel;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace BLL.Services.Chat
{
    public class ChatService : IChatService
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly IMapper _mapper;

        public Task AddChatWithMembersAsync(string title,
List<OutputApplicationUserDTO> applicationUsers)
        {
            throw new NotImplementedException();
        }

        public Task AddMembersToChatAsync(Guid id,
List<OutputApplicationUserDTO> applicationUsers)
        {
            throw new NotImplementedException();
        }

        public Task DeleteChatAsync(Guid id)
        {
            throw new NotImplementedException();
        }

        public Task<InputChatDTO> GetAllChatsByUserId(Guid id)
        {
            throw new NotImplementedException();
        }

        public Task UpdateChatTitleAsync(Guid id, string title)
        {
            throw new NotImplementedException();
        }
    }
}

```

Б.34 Текст файла CommentService.cs

```

using AutoMapper;
using BLL.DTO.Chat.Out;
using BLL.DTO.In;
using BLL.Exeption;
using BLL.Services.Chat.Interfaces;
using DBLevel;
using DBLevel.EF.Entities.Chats;
using DBLevel.Extensions;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BLL.Services.Chat
{
    public class CommentService : ICommentService
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly IMapper _mapper;

        public CommentService(IUnitOfWork unitOfWork, IMapper mapper)
        {
            _unitOfWork = unitOfWork;
            _mapper = mapper;
        }

        public async Task AddAsync(InputCommentDTO model)
        {
            Guard.VerifyNullObject(model, nameof(AddAsync),
this.ToString());

            model.Id = IdGenerator.GenereteId();
            var comment = _mapper.Map<Comment>(model);

            await _unitOfWork.CommentRepository.AddAsync(comment);
            await _unitOfWork.SaveChangesAsync();
        }

        public async Task DeleteByKeyAsync(Guid id)
        {
            Guard.VerifyEmpthyId(id, nameof(DeleteByKeyAsync),
this.ToString());

            await _unitOfWork.CommentRepository.DeleteByIdAsync(id);
            await _unitOfWork.SaveChangesAsync();
        }

        public async Task UpdateAsync(InputCommentDTO model)
        {
            Guard.VerifyNullObject(model, nameof(UpdateAsync),
this.ToString());

            _unitOfWork.CommentRepository.Update(_mapper.Map<Comment>(model));
            await _unitOfWork.SaveChangesAsync();
        }

        public async Task<IEnumerable<OutputCommentDTO>>
        GetAllByChatIdAsync(Guid id)
        {
            var collectionCommetns = await Task.Run(() =>

```

```

        _unitOfWork.CommentRepository.FindAllByChatIdWithDetails(id)
    );

    return _collectionComments.Select(el) =>
_mapper.Map<OutputCommentDTO>(el)).ToList();
    }
}
}

```

Б.35 Текст файла Startup.cs

```

using DBLevel.EF;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Serilog;
using Microsoft.EntityFrameworkCore;
using UIL.Middlewares;
using DBLevel;
using DBLevel.Entities.Abstraction;
using Microsoft.AspNetCore.Identity;
using BLL.Services.AdministrativePanel;
using BLL.Services.Profile.Interfaces;
using BLL.Services.AdministrativePanel.Interfaces;
using BLL.Services.Profile;
using BLL.Services.Order.Interfaces;
using BLL.Services;
using BLL.Services.Chat.Interfaces;
using BLL.Services.Interfaces.Chat;
using BLL.Services.Chat;

namespace FreelanceExchangeSoftware
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

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

            services.AddDbContext<FESContext>(
                options =>
options.UseSqlServer(Configuration.GetConnectionString("FESConnection")));

            services.AddIdentity<ApplicationUser, IdentityRole>(
                options =>
                {
                    options.SignIn.RequireConfirmedAccount = false;
                }
            )
            .AddEntityFrameworkStores<FESContext>();
        }
    }
}

```

```

services.AddScoped<IUnitOfWork, UnitOfWork>();

services.AddScoped<IChatService, ChatService>();
services.AddScoped<ICommentService, CommentService>();
services.AddScoped<IOrderService, OrderService>();
services.AddScoped<IProfileService, ProfileService>();
services.AddScoped<IAdministratorPanelService,
AdministratorPanelService>();

services.AddRazorPages();

services.AddControllers();

services.AddControllersWithViews();

services.AddMvc();

services.AddMemoryCache();

services.AddSingleton((container) =>
{
    var configuration = new ConfigurationBuilder()
        .AddJsonFile($"appsettings.json")
        .Build();

    Log.Logger = new LoggerConfiguration()
        .ReadFrom.Configuration(configuration)
        .CreateLogger();
    return Log.Logger;
});
}

// This method gets called by the runtime. Use this method to
configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment
env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    {
        app.UseExceptionHandler("/Error");
        // The default HSTS value is 30 days. You may want to change
this for production scenarios, see https://aka.ms/aspnetcore-hsts.
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseDefaultFiles();

    app.UseSerilogRequestLogging();

    app.UseRouting();

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

    app.UseMiddleware<GlobalLoggerMiddleware>();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapRazorPages();
    }

```

```

        endpoints.MapControllers();
    });
}
}
}

```

Б.36 Текст файла Program.cs

```

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using Serilog;
using System;
using System.Diagnostics.CodeAnalysis;
using Microsoft.Extensions.DependencyInjection;
using DBLevel.EF;

namespace FreelanceExchangeSoftware
{
    [ExcludeFromCodeCoverage]
    public class Program
    {
        public static void Main(string[] args)
        {
            string environmentName;

            environmentName = "Development";

            var configuration = new ConfigurationBuilder()
                .AddJsonFile($"appsettings.{environmentName}.json")
                .Build();

            Log.Logger = new LoggerConfiguration()
                .ReadFrom.Configuration(configuration)
                .CreateLogger();

            try
            {
                Log.Information("Application Stating Up");
                var host = CreateHostBuilder(args)
                    .UseEnvironment(environmentName)
                    .Build();

                CreateDbIfNotExists(host, Log.Logger);
                host.Run();
            }
            catch (Exception ex)
            {
                Log.Fatal(ex, "The application failed to start correctly");
            }
            finally
            {
                Log.CloseAndFlush();
            }
        }

        private static void CreateDbIfNotExists(IHost host, ILogger logger)
        {
            using (var scope = host.Services.CreateScope())
            {
                var services = scope.ServiceProvider;
                try

```

```

        {
            var context =
services.GetRequiredService<FESContext>();
           _INITIALIZER.Initialize(context);
        }
        catch (Exception ex)
        {
            logger.Fatal(ex, "An error occurred creating the DB.");
        }
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.ConfigureAppConfiguration((hostingContext,
config) =>
                {
                    var env = hostingContext.HostingEnvironment;

                    config.AddJsonFile("appsettings.json", optional:
true, reloadOnChange: true)

.AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true,
reloadOnChange: true);
                });

                webBuilder.UseStartup<Startup>();
            })
            .UseSerilog();
    }
}

```

Б.37 Текст файла CommentController.cs

```

using AutoMapper;
using GameStore.BLL.DTO;
using GameStore.BLL.Interfaces;
using GameStore.UIL.ViewModels.In;
using GameStore.UIL.ViewModels.Out;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace GameStore.WEB.Controllers
{
    [Route("[controller]")]
    public class CommentController : Controller
    {
        private readonly ICommentService _commentService;
        private readonly IMapper _mapper;

        public CommentController(IGameService gameService, ICommentService
commentService, IUserService userService , IMapper mapper )
        {
            _commentService = commentService;
            _mapper = mapper;
        }

        [Route("{chatid}/comments")]
    }
}

```

```

[HttpGet]
public async Task<ViewResult> CommentsAsync(string chatid)
{
    ViewBag.Comments = _mapper.Map<IEnumerable<CommentDTO>,
IEnumerable<CommentViewModel>>
        ( await _commentService.GetAllByGameAsync(chatid));

    ViewBag.gamekey = chatid;
    return View();
}

[Route("{chat}/newcomment")]
[HttpPost]
public async Task<ActionResult> AddCommentForGameAsync([FromForm]
CreateCommentViewModel newcomment, string chat)
{
    await
        _commentService.AddCommentAsync(_mapper.Map<CommentDTO>(newcomment), chat,
        await _userService.FindByNameAsync("IdentityNameAuthor"));
    return RedirectToAction("Comments", "Comment", new { gamekey =
chat });
}
}
}

```

Б.38 Текст файла GlobalLoggerMiddleware.cs

```

using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Routing;
using Serilog;
using System;
using System.Threading.Tasks;

namespace UIL.Middlewares
{
    public class GlobalLoggerMiddleware
    {
        private readonly RequestDelegate _next;
        public GlobalLoggerMiddleware(RequestDelegate next)
        {
            this._next = next;
        }

        public async Task InvokeAsync(HttpContext context)
        {
            try
            {
                Log.Information($"{context.Response.HttpContext.Request}
request processed successfully, status code:
{context.Response.HttpContext.Response.StatusCode}, along the request path"
+
                $"{context.Request.Path} , controller:
{context.GetRouteValue("controller")}, action:
{context.GetRouteValue("action")} ");
                context.Response.StatusCode = StatusCodes.Status200OK;
                await _next.Invoke(context);
            }
            catch (Exception ex)
            {
                context.Response.StatusCode =
                StatusCodes.Status404NotFound;
                Log.Fatal(ex, "The application failed");
            }
        }
    }
}

```

```

        finally
        {
            Log.CloseAndFlush();
        }
    }
}

```

Б.39 Текст файла ChatRepositoryTests.cs

```

using AutoFixture;
using DbLevel.Tests.HelperClasses;
using DBLevel.Repositories.ChatsRepositories;
using System.Threading.Tasks;
using Xunit;

namespace DbLevel.Tests.Repositories.ChatsRepositories
{
    public class ChatRepositoryTests
    {
        private const string ExpectedDescription = "test description";
        private const string Id = "F23A7BCC-FE5D-4763-89C1-CD9FB4CAE1D4";
        private const string ExpectedName = "test Name";
        private readonly CommentRepository _commentRepository;
        private readonly DatabaseFixture _databaseFixture;
        private readonly Fixture _fixture;

        public ChatRepositoryTests()
        {
            _databaseFixture = new DatabaseFixture();
            _commentRepository = new CommentRepository(_databaseFixture.Context);
            _fixture = new Fixture();
        }

        private async Task SeedData()
        {
            var entity = await _commentRepository.GetByIdAsync(new Guid(Id));
            if (entity != null)
            {
                _commentRepository.Delete(entity);
            }
            var author1 = _fixture.Build<Comment>()
                .With(p => p.Description, ExpectedDescription)
                .With(p => p.DateCration, DateTime.UtcNow)
                .With(p => p.Id, new Guid(Id))
                .Create();

            await Task.WhenAll(_commentRepository.AddAsync(author1));
            await _databaseFixture.Context.SaveChangesAsync();
        }

        ~ChatRepositoryTests()
        {
            _databaseFixture.Dispose();
        }

        [Fact]
        public async Task FindAll_WhenIsExistEntity_IEnumerableChat()
        {
            //Arrange
            await SeedData();
        }
    }
}

```

```

var expectedCol = _databaseFixture.Context.Comments.Count();

//Act
var col = _commentRepository.FindAll();
var entity = col.First();

//Assert
Assert.Equal(expectedCol, col.Count());
Assert.Equal(ExpectedDescription, entity.Description);
}

[Fact]
public async Task GetByIdAsync_WhenIdIsExist_Chat()
{
    //Arrange
    await SeedData();
    var id = new Guid(Id);

    //Act
    var entity = await _commentRepository.GetByIdAsync(id);

    //Assert
    Assert.Equal(ExpectedDescription, entity.Description);
}

[Fact]
public async Task AddAsync_NewCommentCreated_Chat()
{
    //Arrange
    var expectedCol = _databaseFixture.Context.Comments.Count() +
1;

    var author = _fixture.Build<Comment>()
        .With(p => p.Description, ExpectedDescription)
        .With(p => p.DateCration, DateTime.UtcNow)
        .With(p => p.Id, new Guid(Id))
        .Create();

    //Act
    await _commentRepository.AddAsync(author);
    _databaseFixture.Context.SaveChanges();
    var col = _commentRepository.FindAll();

    //Assert
    Assert.Equal(expectedCol, col.Count());
    Assert.Equal(ExpectedDescription, col.Last().Description);
}

[Fact]
public async Task Delete_WhenCommentDeleted_Chat()
{
    //Arrange
    var author = _fixture.Create<Comment>();
    await _commentRepository.AddAsync(author);
    _databaseFixture.Context.SaveChanges();
    var expectedCol = _databaseFixture.Context.Comments.Count() -
1;

    //Act
    _commentRepository.Delete(author);
    _databaseFixture.Context.SaveChanges();
    var col = _commentRepository.FindAll();

    //Assert
    Assert.Equal(expectedCol, col.Count());

```

```

    }

    [Fact]
    public async Task Update_WhenEntityIsUpdated_Chat()
    {
        //Arrange
        await SeedData();
        var entity = _databaseFixture.Context.Comments.First();
        entity.Description = ExpectedDescription;

        //Act
        _commentRepository.Update(entity);
        _databaseFixture.Context.SaveChanges();
        var veifyEntity = _databaseFixture.Context.Comments.First();

        //Assert
        Assert.Equal(ExpectedDescription, veifyEntity.Description);
    }
}

```

ДОДАТОК В
Слайди презентації

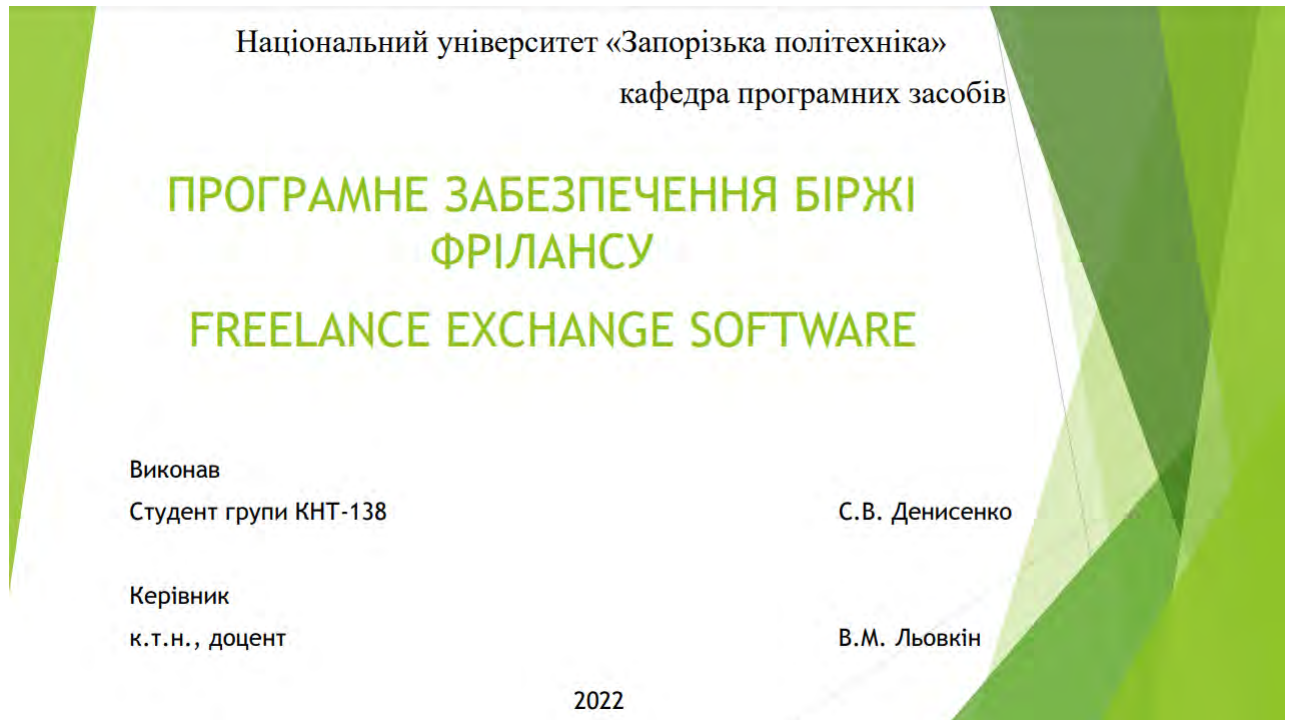


Рисунок В.1 – Слайд № 1

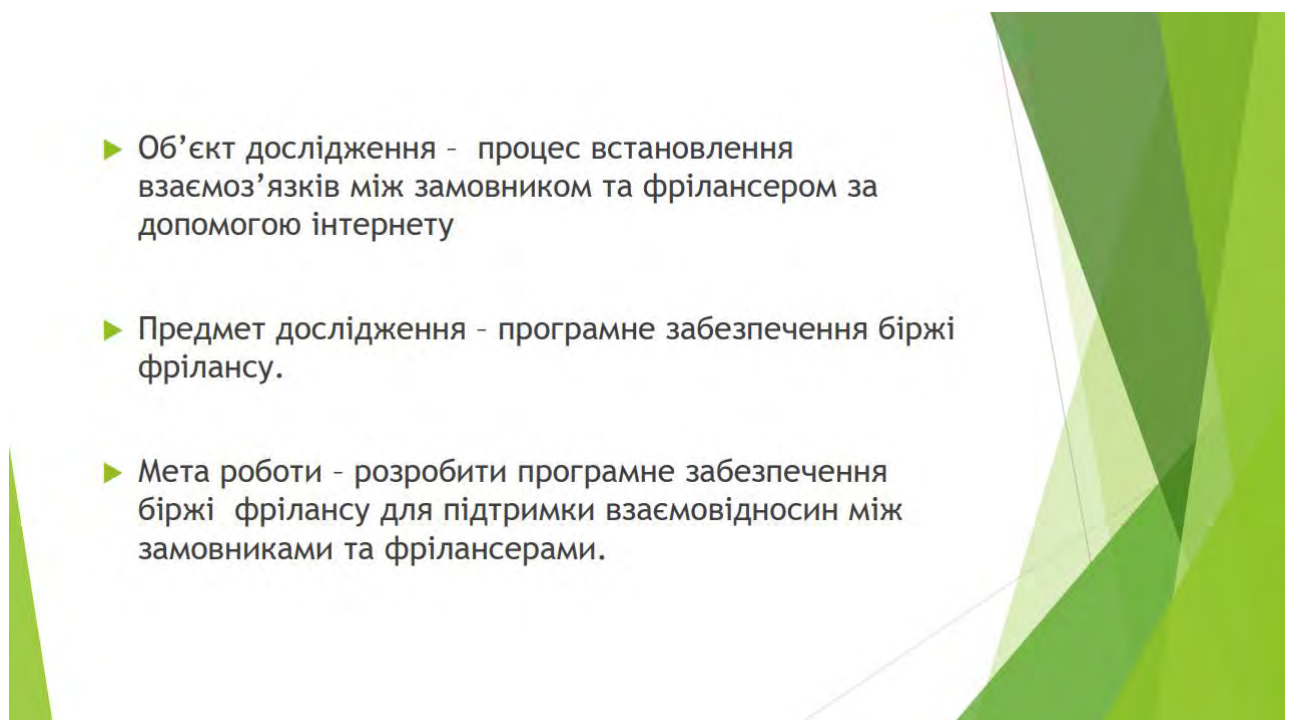


Рисунок В.2 – Слайд № 2

Завдання

- Аналіз предметної області
- Визначення проблемних зон предметної області
- Визначення вимог до програмного забезпечення
- Проектування програмного забезпечення біржі фрілансу
- Реалізація і налагодження серверної частини ПЗ

Рисунок В.3 – Слайд № 3

Порівняння аналогів

Критерій оцінки	Аналоги				
	Upwork	Fiverr	99designs	Service Scape	Власний
1	2	3	4	5	6
Чат	+/-	+/-	-	+/-	+/-
Прикріплювання файлів	+	-	-	+	+
Інтерфейс	+	+/-	+	+	+/-
Конкурси	+	+	-	-	-
Профіль фрілансера	+	-	-	-	+
Регулювання відносин	+	-	-	+	+
Швидка зміна ролі	-	-	-	-	+
Алгоритм	-	+	-	-	-
Пошук фрілансерів	+	-	-	-	+

Рисунок В.4 – Слайд № 4

Порівняння мов програмування

Мова програмування	Критерії оцінки мови					
	Типізація	Швидкість	Кросплатформеність	Вивчення	Кількість інформації	Фреймворки та бібліотеки
1	2	3	4	5	6	7
JavaScript TypeScript	динамічна /слабка динамічна	швидше ніж середня	+	складно	Висока	Велика
Python	динамічна	повільно	+	середня	Висока	Велика
Java	статична	швидко	+	дуже складно	Висока	Середня
C#	нестрога статична	швидше ніж середня	+/-	складно	Висока	Велика
PHP	слабка динамічна	повільно	+	середня	Середня	Середня
Go	строга статична	середня	+	легко	Середня	Невелика

Рисунок В.5 – Слайд № 5

Схема бази даних

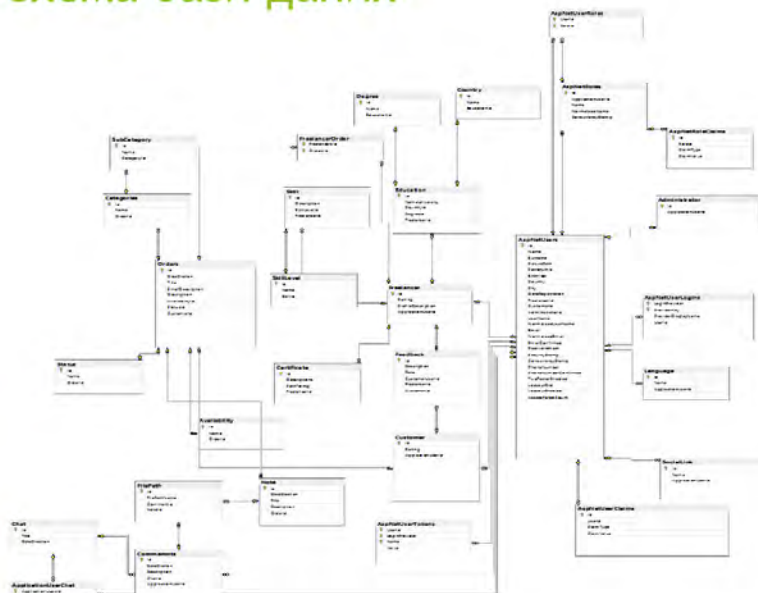


Рисунок В.6 – Слайд № 6

Архітектура веб застосунку

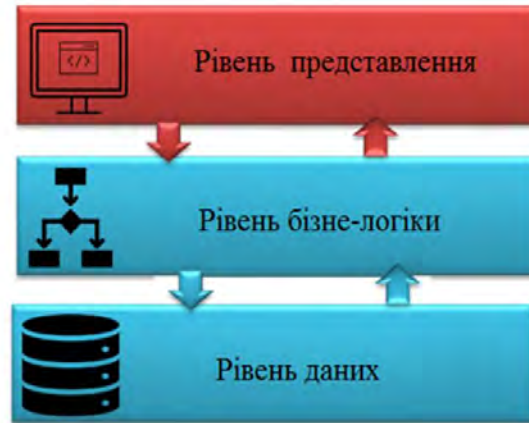


Рисунок В.7 – Слайд № 7

Алгоритм функціонування



Рисунок В.8 – Слайд № 8

Бізнес процес конвеєра запитів

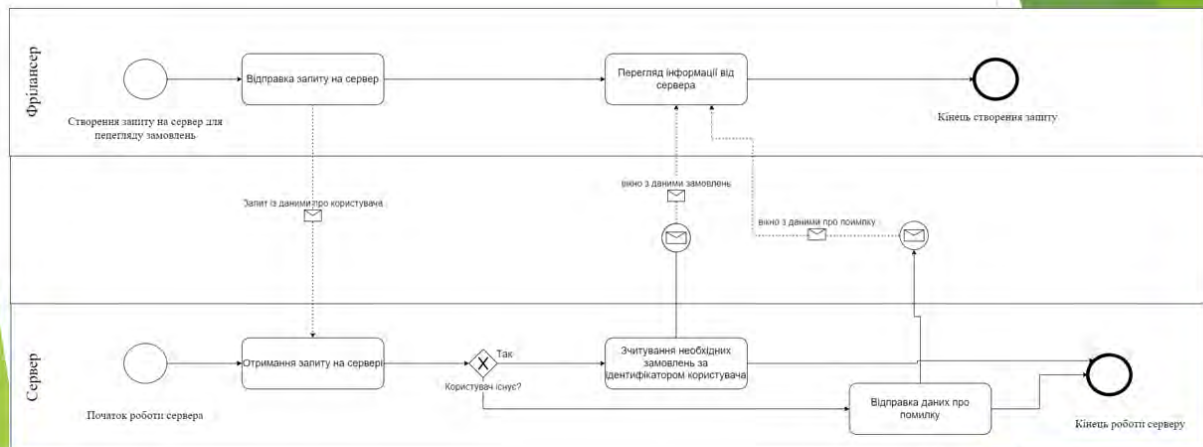


Рисунок В.9 – Слайд № 9

Класи серверного застосунку

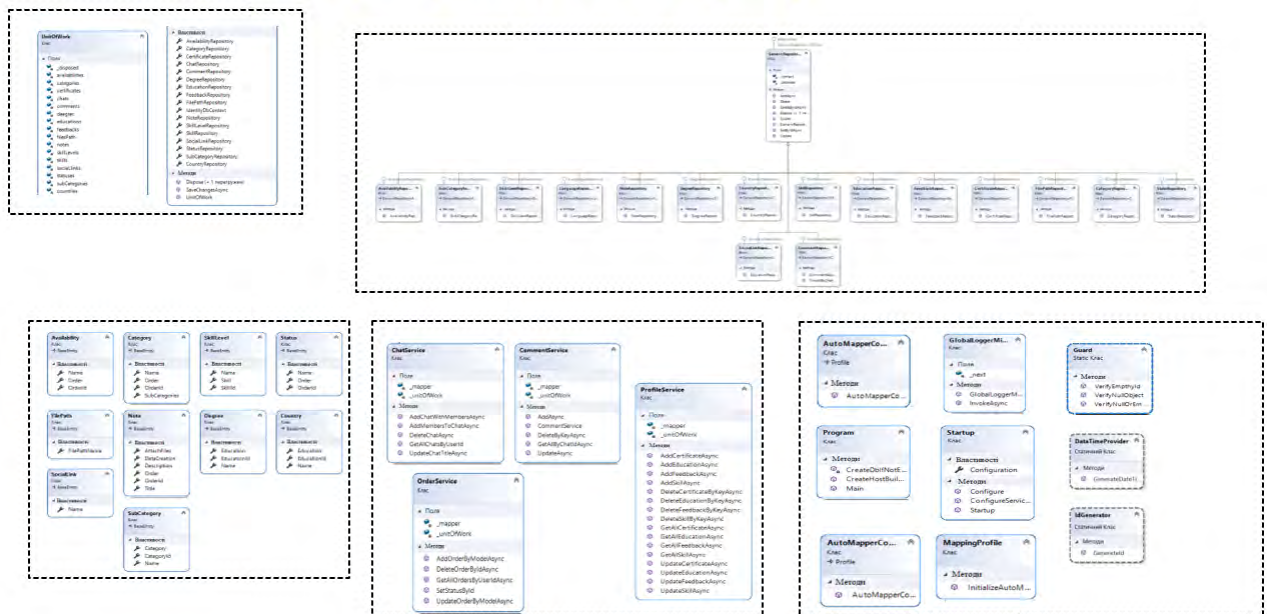


Рисунок В.10 – Слайд № 10

Інтерфейси користувача

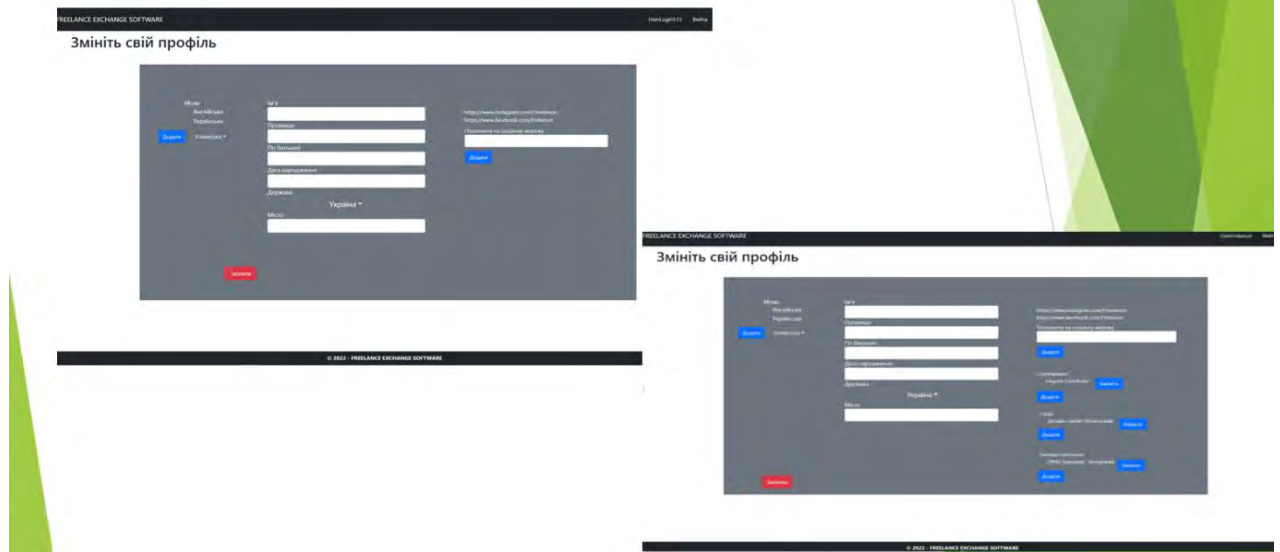


Рисунок В.11 – Слайд № 11

Інтерфейси користувача

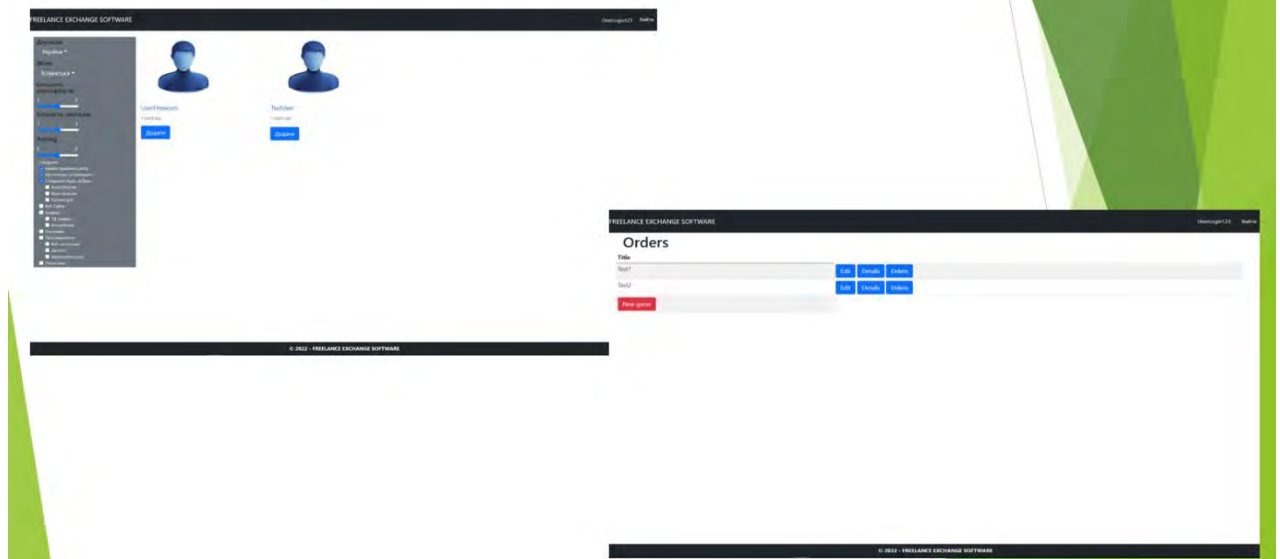


Рисунок В.12 – Слайд № 12

Звіт по тестуванню

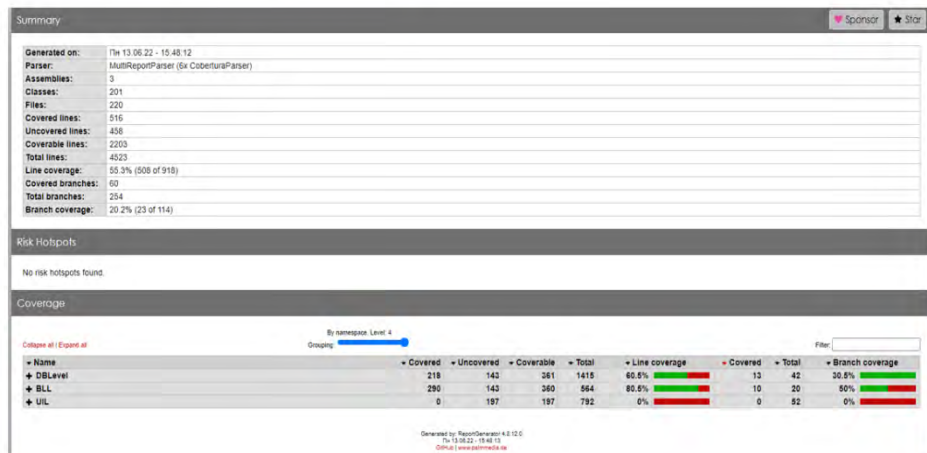


Рисунок В.13 – Слайд № 13

Висновки

- Для реалізації програми було використано мову програмування C# із застосуванням інтегрованого середовища розробки Visual Studio. Була обрана платформа для вебзастосунку WEB API, що у майбутньому дозволить перейти на більш сучасні технології для інтерфейсної частини, такі як React або Vue та інші. У застосунку використовувалася система керування реляційними БД Microsoft SQL Server та SQLite для тестів. Для адміністрування та звітності використовувалася утиліта SQL Server Management Studio. Для роботи з даними був використаний підхід Code First та фреймворк Entity Framework Core, крім цього були використані бібліотеки AutoMapper, XUnit, AutoFixture, Serilog, Identity. Для інтерфейсної частини був використаний Bootstrap, мови HTML і CSS, а також фреймворк MVC та генератор сторінок Razor. Був використаний багатoshаровий архітектурний шаблон. Також у застосунку були застосовані принципи SOLID, dependency inversion та dependency injection.
- Були представлені базові інтерфейсні вікна для роботи користувача з програмою. Було виконано тестування програми за допомогою модульного тестування, а також загалом визначено покриття тестами.
- Програма виконана у вигляді серверного застосунку з використанням клієнт-серверної архітектури, де клієнтом виступає браузер, а сервером розроблюваний застосунок.
- Серверний застосунок розроблений для підтримки людей (фрілансерів), у їх прагненні працювати та займатися саморозвитком, також застосунок прагне надати рівний доступ фрілансерам за допомогою більшої кількості фільтрів, щоб замовник міг з легкістю вибрати саме потрібного фрілансера. Також була додана функція швидкої зміни ролі для користувачів. У майбутньому цей застосунок можна розширити та додати функціональність для вивчення фрілансером популярних платформ саморозвитку, запровадити функціональність для існування великих компаній у цій сфері, а також вирішити правову проблему, яка досить часто зустрічається у даній області.

Рисунок В.14 – Слайд № 14