МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА СИСТЕМИ КЕРУВАННЯ ВЗАЄМОДІЄЮ КЛІЄНТАМИ З БЮРО ПЕРЕКЛАДУ»**

Виконав: студент 2 курсу, групи 8.1221

спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)

освітньої програми комп'ютерні науки
(назва освітньої програми)

Д.А. Шкрьоба
(ініціали та прізвище)

Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри інформаційних технологій в туризмі НУЗП, доцент, к.т.н. Морозов Д.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя
2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
«ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти магістр

Спеціальність 122 комп'ютерні науки

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук, д.т.н., професор

_____ Чопоров С.В

(підпис)

« ____ » _____ 2022 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Шкрьобі Данилу Андрійовичу

(прізвище, ім'я та по– батькові)

1. Тема роботи (проекту) Розробка системи керування взаємодією клієнтами з бюро перекладу

керівник роботи (проекту) Решевська Катерина Сергіївна, к.т.н., доцент

(прізвище, ім'я та по– батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 05 » 05 2022 року № 500-с

2. Строк подання студентом роботи

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково– пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Проектування програмного доповнення.
3. Реалізація та тестування програмного доповнення.

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1			
2			
3			
Додатки			

7.Дата видачі завдання 05.05.2022

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану виконання кваліфікаційної роботи магістра.		
2.	Збір вихідних даних та аналіз предметної області.		
3.	Обробка методичних та теоретичних джерел.		
4.	Специфікація вимог до системи. Робота над першим розділом.		
5.	Проектування системи. Робота над другим розділом.		
6.	Реалізація та тестування системи. Робота над третім розділом.		
7.	Розробка керівництва користувача. Робота над додатками.		
8.	Оформлення та нормоконтроль кваліфікаційної роботи.		
9.	Захист кваліфікаційної роботи.		

Студент _____ Д.А. Шкрьоба
(підпис) (ініціали та прізвище)

Керівник роботи _____ К.С. Решевська
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка системи керування взаємодією клієнтами з бюро перекладу»: 73 с., 32 рис., 1 табл., 12 джерел, 5 додатків.

ANGULAR, API, FRAMEWORK, LARAVEL, MVC, PHP, TYPESCRIPT, UML.

Об’єкт дослідження – система, інструменти для взаємодії Angular та Laravel, засоби взаємодії системи з користувачем.

Предмет дослідження – засоби взаємодії системи замовлення перекладів з користувачем.

Мета роботи – розробити систему для замовлення перекладів.

Методи дослідження – моделювання, проектування, програмний, аналітичний.

У епоху розвитку інформаційних технологій значно спростився процес укладання угод, як компаніями, так і окремими персонами. Але документи, які супроводжують угоду можуть бути чужій для однієї із сторін мові. У зв’язку з цим велику актуальність набуває розробка систем, які дозволяють швидко замовити якісний переклад.

Як правило, подібні системи мають різний функціонал, який охоплює різні аспекти обробки та супроводу перекладів. Але відсутність або обмеженість потрібного функціоналу не є виключенням. Одним з прикладів такої проблеми є обмеженість роботи компанії з деякими видами документів та незручний інтерфейс користувача. Таким чином, за результатами роботи створено зручну та ефективну систему замовлення перекладів з використанням Angular та Laravel.

SUMMARY

Master’s qualifying paper «Development of the Customer Relationship Management System for a Translation Agencv»: 73 pages, 32 figures, 1 table, 12 sources, 5 appendices.

ANGULAR, API, FRAMEWORK, LARAVEL, MVC, PHP, TYPESCRIPT, UML.

The object of the study is the system, Angular and Laravel interaction tools, means of interaction between the system and the user.

The subject of the study is the means of interaction of the translation ordering system with the user.

The aim of the work is to develop a system for ordering translations.

The methods of research are modeling, design, software, analytical.

In the era of information technology development, the process of concluding agreements, both by companies and individuals, has become much easier. But the documents accompanying the agreement may be in a language foreign to one of the parties. In this regard, the development of systems that allow you to quickly order a high-quality translation is gaining great relevance.

As a rule, such systems have different functionality that covers different aspects of translation processing and support. But the absence or limitation of the required functionality is not an exception. One example of such a problem is the limitation of the company’s work with some types of documents and an inconvenient user interface. Thus, based on the results of the work, a convenient and effective translation ordering system using Angular and Laravel was created.

ЗМІСТ

ВСТУП

У епоху розвитку інформаційних технологій значно спростився процес укладання угод, як компаніями, так і окремими персонами. Але документи, які супроводжують угоду можуть бути чужій для однієї із сторін мові.

Системи для перекладу текстів та документів стали невід'ємною частиною в забезпеченні функціоналу щодо супроводу угод між юридичними та/або фізичними особами. Знайдеться небагато людей, котрі б не користувалися послугами таких систем. За допомогою них клієнти мають можливість замовити переклад різних типів документів, від технічної документації до юридичних договорів. У зв'язку з цим велику актуальність набуває розробка систем, які дозволяють швидко замовити якісний переклад.

Отож, які шляхи вирішення даної проблеми:

– самостійно створити систему, яка буде мати необхідний функціонал;
– користуватися перевіреними сервісами для кожної групи документів.

Але великою проблемою є обмеженість роботи компанії з деякими видами документів та незручний інтерфейс користувача. Також проблемою є можливість закриття сервісу або призупинення роботи з деякими категоріями документів, у цьому разі користувачу необхідно шукати новий сервіс. Виходячи з цього, було вирішено створити систему, котра би мала розширений функціонал і була доступна усім бажаючим.

Актуальність дослідження: актуальність теми зумовлена необхідністю перекладу документів для різних сфер діяльності.

З огляду на це, можна виділити наступні цілі і задачі нашого дослідження:

Мета: розробити систему замовлення перекладів.

Задачі:

– сформувати вимоги до системи;
– спроектувати та побудувати архітектуру системи;

– реалізувати систему замовлення перекладів;

– протестувати роботу системи.

Об’єкт дослідження: процес замовлення перекладу користувачем, інструменти для реалізації системи замовлення перекладу, функціонал системи замовлення перекладу, необхідний користувачам.

Предмет дослідження: створення системи, що дозволяє замовляти переклад різних категорій документів.

Методи дослідження: моделювання, проектування, програмний, аналітичний.

Перший розділ присвячено збору та аналізуванню вимог до системи, огляду інструментів розробки та опису системи.

У другому розділі розглянуто етапи проектування системи, наведено детальний опис прецедентів.

Третій розділ присвячено реалізації та тестуванню роботи системи, наведено детальне керівництво користувача, описующе процес роботи у системі.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Порівняння з подібними системами

Перше питання, котре можно задати: “Навіщо створювати такий сервіс, якщо вже існують подібні?”. Так, таких сервісів створено багато, але якість багатьох з них залишає бажати кращого.

Так як ми не можемо порівняти якість перекладу, будемо порівнювати сервіси спираючись на їх веб-сайти у мережі. Критерії для порівняння: зрозумілість сайту, процес замовлення, відгуки, наявність прайсу, різноманітність послуг.

Пройдемося по кожному пункту окремо:

- зрозумілість сайту – наскільки на сайті можна швидко знайти потрібні розділи;
- процес замовлення – наскільки простий і зрозумілий, не містить специфічних термінів і назв без пояснення;
- відгуки – важлива частина подібного сервісу, допомагає зрозуміти якість послуг;
- наявність прайсу – повинен включати до себе всі можливі аспекти формування вартості замовлення;
- різноманітність послуг – спектр послуг, які надає сервіс.

Для порівняння були обрані сервіси: “Антарио М”, “УКРПЕРЕВОД”, “Mova Club”, документ-сервис «ГОТОВО!». Порівняння з нашою системою наведені у таблиці 1.1.

Таблиця 1.1 – Порівняння сервісів перекладу

	Зрозумілість сайту	Процес замовлення	Відгуки	Наявність прайсу	Різноманітність послуг
Антарио М	Поганий дизайн сайту не дає можливості одразу знайти потрібні розділи. Автоматично відкривається вікно чату, що закриває частину контенту.	Відсутній. Сайт носить інформативний характер.	Присутні.	Присутній, покриває всі аспекти замовлення.	Торкається різних сфер життєдіяльності (медицинські, юридичні, технічні, тощо). Є послуга нотаріальної завірки.
УКРПЕРЕВОД	Зрозумілий дизайн. Послідовна структура сторінки.	Плюси: простий процес, який не потребує реєстрації. Мінуси: клієнт не бачить фінальну вартість замовлення.	Присутні	Присутній, покриває всі аспекти замовлення.	Торкається різних сфер життєдіяльності (медицинські, юридичні, технічні, тощо). Є послуга нотаріальної завірки.
Mova Club	Зрозумілий дизайн. Послідовна структура сторінки.	Плюси: простий процес, який не потребує реєстрації. Мінуси: клієнт не бачить фінальну вартість замовлення.	Є тільки рекомендації	Відсутні	Торкається різних сфер життєдіяльності (медицинські, юридичні, технічні, тощо). Є послуга нотаріальної завірки.

Продовження таблиці 1.1

ГОТОВО!	Зрозумілий дизайн. Послідовна структура сторінки. Автоматично відкривається вікно чату, що закриває частину контенту.	Відсутній. Сайт носить інформативний характер.	Відсутні	Відсутні	Торкається не всіх сфер життєдіяльності. Є послуга нотаріальної завірки.

Як бачимо, є функції, які є у одних сервісів, але немає у інших. Наша система покликана об’єднати кращі аспекти розглянутих сервісів та покращити негативні.

2 ТЕХНІЧНЕ ЗАВДАННЯ

2.1 Терміни та визначення

2.1.1 Загальні терміни

Система – веб-сайт створений на основі Angular та Laravel.

Angular – JavaScript-фреймворк з відкритим програмним кодом. Призначений для розробки односторінкових додатків, що складаються з одної HTML сторінки з CSS і JavaScript..

Laravel – PHP-фреймворк, призначений для розробки веб-додатків відповідно до шаблону model–view–controller.

MySQL – це система управління базами даних, яка використовується для підтримки реляційних баз даних.

ВВ – Варіант Використання чи Use Case.

ДВІ – Діаграма Варіантів Використання чи Use Case Diagram.

ДД – Діаграма Діяльності.

ДП – Діаграма Послідовності.

Перекладач – людина, котра зареєстрована у системі та має певні права доступу.

Адміністратор – користувач, котрий має права на редагування системи.

Клієнт – користувач, котрий має права на обмежене коло взаємодії з системою.

2.1.2 Технічні терміни

ІС – інформаційна система.

БД – база даних, місце збереження інформації ІС.

MVC – архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

2.2 Функціональні вимоги

2.2.1 Призначення і цілі створення системи

Функціональне призначення системи – реалізувати можливість клієнтам швидко замовляти переклад.

Експлуатаційне призначення системи: система може експлуатуватися адміністратором і клієнтом системи на різних рівнях доступу.

Мета створення системи – розробка системи для надання послуг перекладу текстів та документів.

2.2.2 Загальні функціональні можливості системи

Система має надавати користувачам такі можливості:

- створення замовлення;
- ввід/редагування даних замовлення;
- завантаження файлів.

Система має надавати адміністраторам такі можливості: редагування/перегляд/видалення замовлень;

- змінювати статус замовлення;
- вхід/вихід до/з системи;
- налаштування системи;
- створення/редагування/перегляд/видалення перекладачів;
- перегляд статистики.

Система має надавати клієнтам такі можливості:

- залишення коментаря;

– оцінювання замовлення.

2.3 Нефункціональні вимоги

2.3.1 Інтерфейс користувача

Система повинна відображати коректно інтерфейс користувача на будь-якому пристрої.

2.3.2 Підтримка браузерів

Система повинна працювати для наступних браузерів останніх версій:

– Mozilla Firefox;
– Google Chrome;
– Safari;
– Opera.

2.3.3 Вимоги до продуктивності

Система повинна відображати будь-яку сторінку не довше, ніж за 3 секунди.

Система повинна відправляти повідомлення не довше, ніж 5 секунд.

2.3.4 Вимоги до безпеки

Система не повинна дозволяти не адміністраторам фізичний доступ до інтерфейсу адміністратора.

Система повинна надавати доступ залежно від прав користувачів.

2.4 Опис предметної області

Предметною областю є розробка системи для надання послуг перекладу текстів та документів. Дана система повинна надати клієнтам можливість швидко замовляти переклад. Процес взаємодії з системою проходить наступним чином. Клієнт, повинен ввести адресу сайту в браузері та обрати “Замовити переклад”.

Система перенаправить клієнта до інтерфейсу замовлення перекладу.

Процесс внесения оформлення замовлення проходить наступним чином. Клієнт обирає файл для завантаження, система відображає статус про успішне завантаження або повідомлення про помилку. Якщо завантаження було успішне, то клієнт повинен натиснути кнопку “Далі”, система відображає форму для вибору мови оригіналу, мови перекладу, типу перекладу та додаткових послуг. Після цього клієнт натискає кнопку “Далі”, система відображає інформацію про замовлення, а також просить ввести адресу електронної пошти та бажану дату виконання перекладу. Після введення усіх даних, якщо клієнт згоден з умовами, то він повинен натиснути кнопку “Оформити замовлення”.

Окрім основних функцій система надає адміністраторам можливість керувати обліковими записами користувачів та переглядати детальну статистику за обраними налаштуваннями.

2.5 Опис системи

У епоху розвитку інформаційних технологій значно спростився процес укладання угод, як компаніями, так і окремими персонами. Але документи, які супроводжують угоду можуть бути чужій для однієї із сторін мові. У зв’язку

з цим велику актуальність набуває розробка систем, які дозволяють швидко замовити якісний переклад.

Системи для перекладу текстів та документів стали невід'ємною частиною в забезпеченні функціоналу щодо супроводу угод між юридичними та/або фізичними особами. Знайдеться небагато людей, котрі б не користувалися послугами таких систем. За допомогою них клієнти мають можливість замовити переклад різних типів документів, від технічної документації до юридичних договорів.

Однією з таких систем є «Бюро перекладів». Нова система для ведення замовлення перекладів.

Можливості системи:

- сторона клієнта:
  - створення замовленнь;
  - створення коментарів.
- консоль адміністратора:
  - управління перекладачами;
  - управління замовленнями;
  - формування звітів;
  - детальна статистика;
  - управління особистими даними.

2.6 Огляд інструментів розробки

2.6.1 Angular

Написаний на TypeScript front-end фреймворк з відкритим кодом, який розробляється під керівництвом Angular Team у компанії Google, а також спільнотою приватних розробників та корпорацій. Angular – це AngularJS,

який переосмислили та який був повністю переписаний тією ж командою розробників.

Angular представляє фреймворк для створення клієнтських додатків. Перш за все він націлений на розробку SPA-рішень (Single Page Application), тобто односторінкових додатків. В цьому плані Angular є спадкоємцем іншого фреймворка AngularJS. У той же час Angular це не нова версія AngularJS, а принципово новий фреймворк.

Архітектура додатка на Angular. Основними елементами в розробці є модулі, компоненти, шаблони, метадані, біндінг даних, директиви, сервіси та ін’єкції залежностей:

- Angular не використовує концепцію "області видимості" або контролерів, натомість як головну архітектурну концепцію він застосовує ієрархію компонентів;
- Angular має інакший синтаксис написання виразів, застосовуючи"[ ]"для біндінгу даних властивостей, і "( )"для біндінгу даних івентів;
- Модульність – значна частина основного функціоналу перенесена у модулі;
- Angular рекомендує та застосовує розроблену Microsoft мову – TypeScript, що містить такі можливості, як:
  - класи, а отже Об’єктно-орієнтоване програмування;
  - система типізації;
  - узагальнене програмування.
- TypeScript – надмножина ECMAScript 6 (ES6), і є зворотно сумісним зі стандартом ECMAScript 5 (тобто JavaScript).Angular також має такі ES6-можливості, як:
  - анонімні функції;
  - ітератори;
  - цикли типу For/Of;
  - python-подібні генератори;

  - рефлексія.
- динамічне завантаження;
- асинхронна компіляція шаблонів;
- заміна контролерів та$scope(області видимості) компонентами та директивами – компонент є директивою з шаблоном;
- ітеративні колбеки завдяки використанню RxJS. RxJS дещо обмежує видимість станів та можливості дебагінгу, але, застосовуючи такі плагіни, як ngReact та ngrx, це легко вирішується.

2.6.2 Laravel

Laravel – це фреймворк для web-додатків з виразним і елегантним синтаксисом. Він дозволить спростити вирішення основних актуальних завдань, таких як аутентифікація, маршрутизація, сесії і кешування. Laravel – це спроба об’єднати все найкраще, що є в інших PHP фреймворках. Основні переваги Laravel:

- велика екосистема з миттєвим розгортанням своєї платформи. Офіційний сайт надає безліч мануалів і інформації для ознайомлення;
- документація Laravel близька до досконалості;
- у Laravel є свій движок для шаблонів Blade, «красивий» синтаксис мови, який сприяє вирішення всіх необхідних завдань, таких як аутентифікація, сесії, кешування і маршрутизація RESTful.

Можливості Laravel:

- пакети – дозволяють створювати і підключати модулі в форматі Composer до додатка на Laravel. Багато додаткові можливості вже доступні у вигляді таких модулів.
- eloquent ORM – реалізація шаблону проектування ActiveRecord на PHP. Дає змогу чітко визначити відношення між об’єктами бази

даних. Стандартний для Laravel побудовник запитів Fluent підтримується ядром Eloquent.

- логіка програми – частина розроблювального додатка, оголошена або за допомогою контролерів, або маршрутів.
- зворотна маршрутизація пов’язує між собою генеруються додатком посилання і маршрути, дозволяючи змінювати останні з автоматичним оновленням пов’язаних посилань. При створенні посилань з допомогою іменованих маршрутів Laravel автоматично генерує кінцеві URL.
- REST-контролери – додатковий шар для розділення логіки обробки GET і POST-HTTP запитів.
- автозавантаження класів – механізм автоматичного завантаження класів PHP без необхідності підключати файли їх визначень у include. Завантаження на вимогу запобігає завантаження непотрібних компонентів; завантажуються тільки ті з них, які дійсно використовуються.
- укладачі уявлень – блоки коду, які виконуються при генерації подання (шаблону).
- ынверсія управління – дозволяє отримувати примірники об’єктів за принципом зворотного керування. Також може використовуватися для створення і отримання об’єктів-одинаків.
- міграції – система керування версіями для баз даних. Дозволяє пов’язувати зміни в коді програми зі змінами, які потрібно внести в структуру БД, що спрощує розгортання та оновлення програми.
- модульне тестування (юніт-тести) – грає дуже велику роль в Laravel, який сам по собі містить велику кількість тестів для запобігання помилок.

2.6.3 MySQL

MySQL – це система управління базами даних, яка використовується для підтримки реляційних баз даних.

MySQL – це масштабована, швидка і надійна система управління базами даних, яка може працювати на будь-якій платформі, такі як Windows, Unix, Linux і т.д., і може бути встановлена на робочому столі або будь-якій серверній машині. Це також дуже добре освоїти порівняно з іншими системами управління базами даних, існуючими на ринку, такими як Microsoft SQL Server, Oracle Database і т.д., і PHP. LAMP – це платформа для веб-розробки, що використовує Linux як операційну систему, веб-сервер apache, реляційну систему управління базами даних mysql та об'єктно-орієнтований сценарій PHP.

У мережевому середовищі клієнт-сервер сервер MySQL доступний як окрема програма. Крім того, він доступний у вигляді бібліотеки, яку можна пов'язати з окремим додатком. Існує декілька корисних програм, що підтримують адміністрування бази даних MySQL. З іншого боку, клієнти mysql встановлюються на комп'ютерах у мережі. Інструкції надсилаються від клієнта mysql на сервер mysql і тоді сервер mysql діє на нього відповідно. Незважаючи на те, що mysql встановлений на одній машині, він здатний надсилати бази даних в декілька локацій, і користувачі можуть отримати доступ до одних і тих же за допомогою різних клієнтських інтерфейсів MySQL. Результати відображаються, коли ці інтерфейси передають оператори SQL серверам. Немає потреби в навчанні нових команд, оскільки функцією mysql можна керувати лише за допомогою існуючих команд SQL. Реплікація даних та розподіл таблиць також можна виконати в mysql, що дозволяє користувачам мати кращу продуктивність та більшу довговічність. Для зберігання та доступу до даних можна використовувати декілька двигунів зберігання даних, як NDB, InnoDB тощо. MySQL написаний на C і C ++ і доступний на численних платформах, включаючи Windows, Linux, Mac та

інші. Ця реляційна система управління базами даних підтримує мільйони записів у базах даних та багатьох типів даних, таких як безпідписані та підписані цілі числа до 8 байт, двійкові, варчар, подвійний, char, float, час, blob, enum, текст, дата, рік, дата, часова мітка, варбінарні та просторові типи OpenGIS. Також підтримуються типи рядків з фіксованою та змінною довжиною. Mysql може підключитися до сервера mysql, використовуючи безліч протоколів, таких як TCP / IP тощо. Поряд з цим він також підтримує кілька інструментів адміністрування клієнтів та утиліт, таких як MySQL Workbench та кілька програм командного рядка.

3 ПРОЕКТУВАННЯ

3.1 Використання UML під час розробки системи

Для розробки системи використовуються різні методології. Моделі дозволяють нам наочно продемонструвати бажану структуру і поведінку системи. Одним з найбільш поширених мов моделювання є уніфікована мова моделювання UML (Unified Modeling Language), призначена для моделювання, уявлення, проектування та документування програмних систем, організаційно-економічних систем, технічних систем та інших систем різної природи. Візуальні моделі дозволяють організувати спілкування між замовниками і розробниками.

UML також використовують для моделювання бізнес-процесів, системного проектування та відображення організаційних структур. UML дозволяє розробникам програмного забезпечення досягти угоди в графічних позначеннях для представлення загальних понять (таких як клас, компонент, узагальнення (generalization), об’єднання (aggregation) і поведінку) і більше сконцентруватися на проектуванні та архітектури. Він спрощує складний процес проектування ПО шляхом створення «креслення» для побудови системи. Мова UML не прив’язана до якої-небудь окремої платформі або мови програмування, тому вона добре підходить для з’єднання мереж різних систем. Вона розроблялася з урахуванням гнучкості і тому здатна адаптуватися до виникаючих нових проблем.

UML містить стандартний набір діаграм і нотацій різноманітних видів. UML містить наступний набір діаграм:

- структурні (structural) моделі:
- діаграма класів (class diagrams) – для моделювання статичної структури класів системи і зв’язків між ними;
- діаграми компонентів (component diagrams) – для моделювання ієрархії компонентів (підсистем) системи;

- діаграми розміщення (deployment diagrams) – для моделювання фізичної архітектури системи;
- моделі поведінки (behavioral):
- діаграми варіантів використання (use case diagrams) – для моделювання функціональних вимог до системи (у вигляді сценаріїв взаємодії користувачів з системою);
- діаграми взаємодії (interaction diagrams), такі як діаграми послідовності (sequence diagrams) і кооперативні діаграми (collaboration diagrams) – для моделювання процесу обміну повідомленнями між об’єктами;
- діаграми станів (statechart diagrams) – для моделювання поводження об’єктів системи при переході з одного стану в інший;
- діаграми діяльності (activity diagrams) – для моделювання поводження системи в рамках різних варіантів використання, або потоків управління.

3.2 Діаграма прецедентів

Для опису функціональної структури системи використовується діаграма варіантів використання. Цей тип діаграм описує загальну функціональність системи. Вони відображають взаємодію між варіантами використання, які представляють функції системи, і діючими особами, які представляють людей або системи, які отримують або передають інформацію в дану систему. Кожна функціональність зображується у вигляді «прецедентів використання» (use case) або просто прецедентів. Прецедент – це типове взаємодія користувача з системою, яка може представляти різні рівні деталізації, описувати видиму користувачем функцію, забезпечувати досягнення конкретної мети, важливої для користувача. Прецедент зображується як овал, пов’язаний з типовими користувачами, які називаються

акторами (actors). Список всіх прецедентів фактично визначає функціональні вимоги до системи.

Призначення даної діаграми складається в наступному: проектовані бізнес-процеси представляється в формі так званих варіантів використання, з якими взаємодіють зовнішні сутності або актори. При цьому актором називається будь-який об’єкт, суб’єкт або система, що взаємодіє з модельованої компанією ззовні. Це може бути людина, технічний пристрій, програма або система, яка служить джерелом впливу на моделюються бізнес процеси систему так, як визначить розробник. Варіант використання служить для опису сервісів і функцій, які компанія надає замовнику. Іншими словами кожен варіант використання визначає набір дій, який чинять компаній при роботі із замовником – актором. При цьому нічого не говориться про те, яким чином буде реалізовано взаємодію акторів з компанією і власне виконання варіантів використання.

Переваги моделі варіантів використання полягають у тому, що вона:

- визначає користувачів і кордони системи;
- визначає системний інтерфейс;
- зручна для спілкування користувачів з розробниками;
- використовується для написання тестів;
- є основою для написання документації користувача.

Між користувачами і варіантами використання можуть бути різні види зв’язку. Основні види зв’язку наступні:

- розширення (extend) – показує, що варіант використання розширює базову послідовність дій і вставляє власну послідовність. При цьому на відміну від типу відносин «включення» розширена послідовність може здійснюватися в залежності від певних умов.
- включення (include) – показує, що варіант використання включається в базову послідовність і виконується завжди. На рисунку 3.1 представлена діаграма прецедентів.

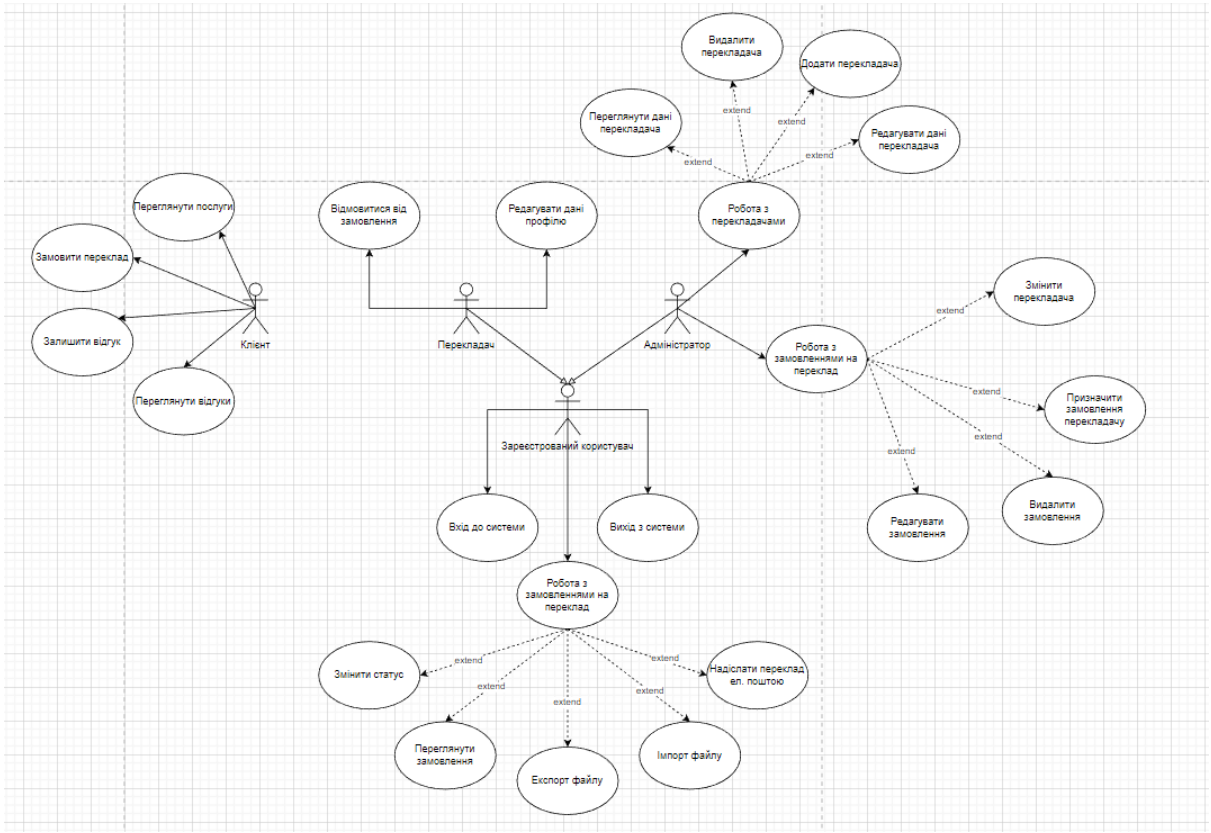


Рисунок 3.1 – Діаграма прецедентів

На діаграмі представлені актори: адміністратор, клієнт, який безпосередньо є користувачем системи з боку «клієнт» та перекладач. Також представлений актор «Зареєстрований користувач», який відображає спільні функціональні можливості адміністратора та перекладача.

Виділено 1 основний варіант використання – «замовити переклад». Після того, як клієнт переходить на головну сторінку сайту та натискає кнопку «Замовити переклад». Система відображає форму створення замовлення. Після введення валідних даних та прикріплення файлу для перекладу, система відображає інформацію про замовлення. Клієнт натискає кнопку «Оформити замовлення», система зберігає замовлення.

Варіанти використання визначають функціональні можливості. Кожен з них представляє певний спосіб використання. Таким чином, кожен варіант використання відповідає послідовності дій для того, щоб клієнт міг отримати певний результат. (див. рис. 3.2 – 3.4).

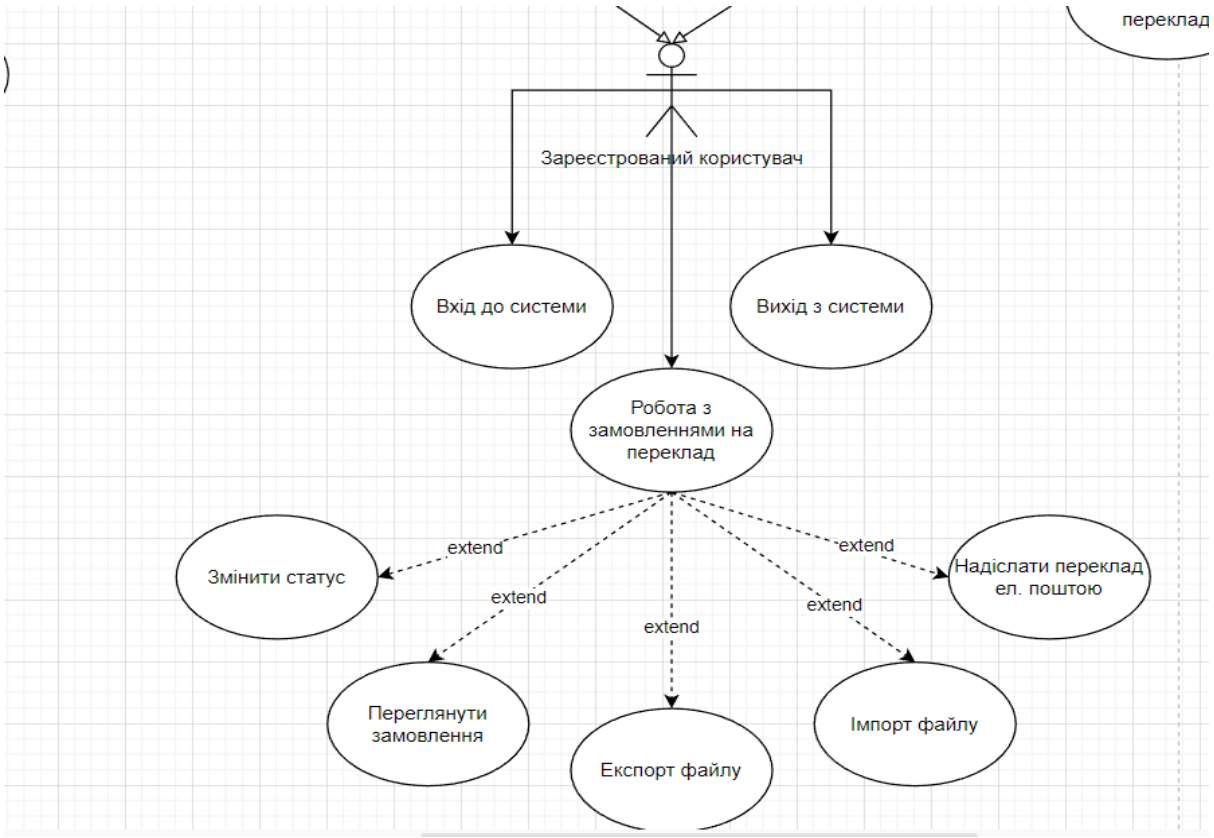


Рисунок 3.2 – Діаграма варіантів використання Зареєстрований користувач

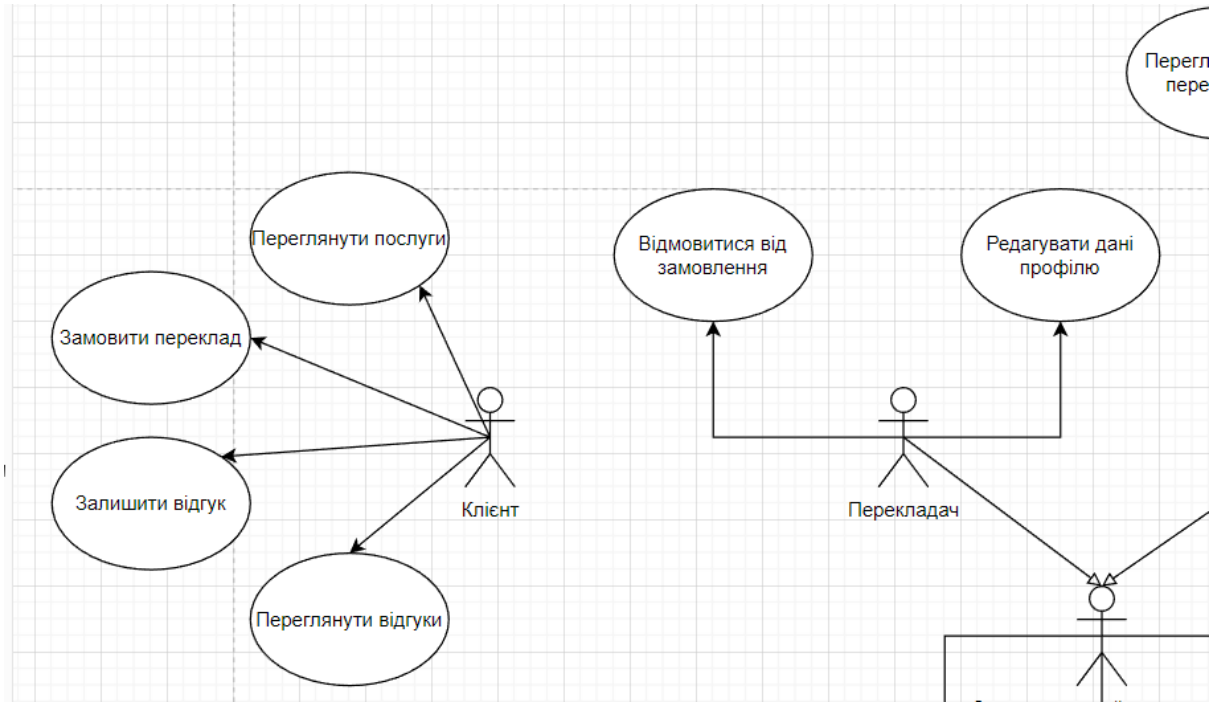


Рисунок 3.3 – Діаграма варіантів використання Клієнт та Перекладач

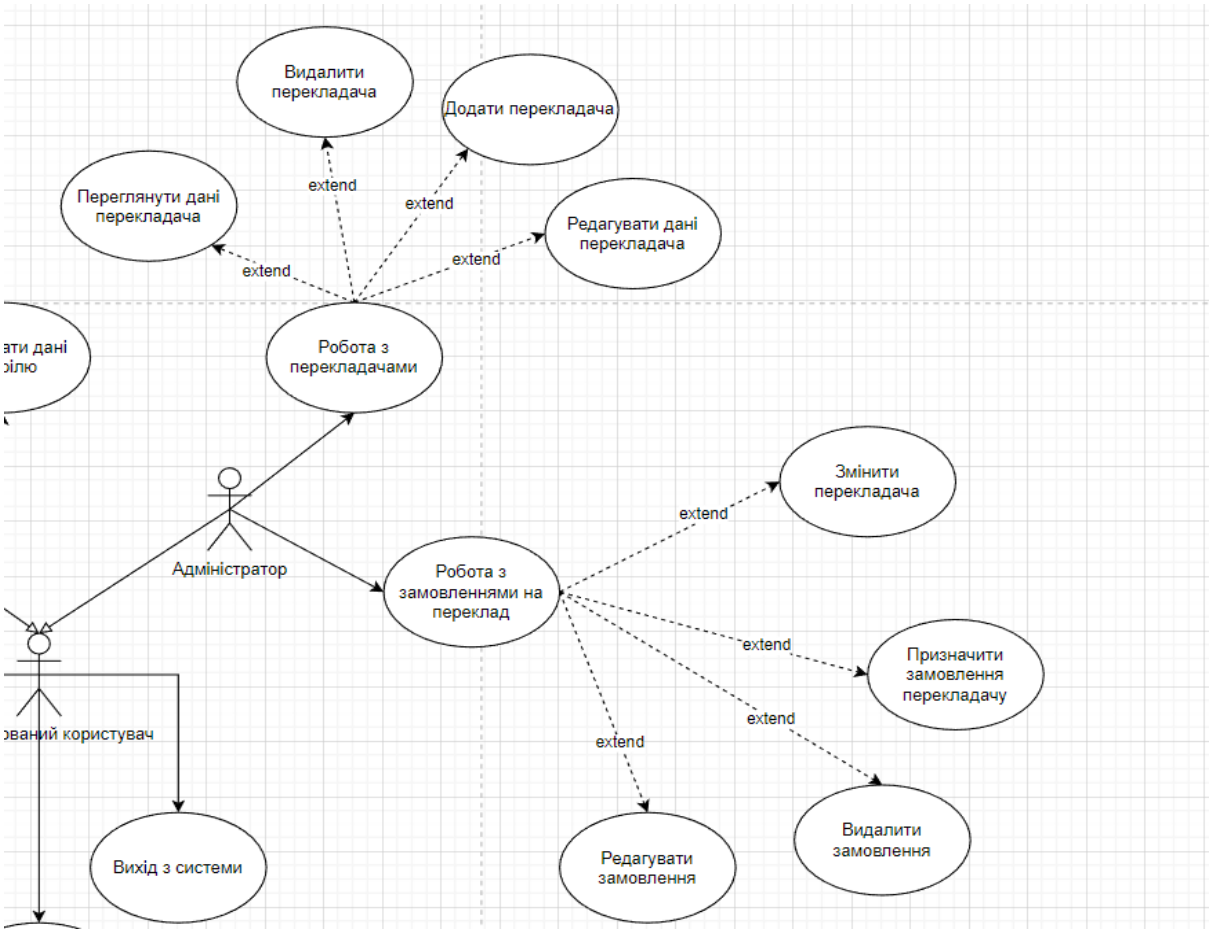


Рисунок 3.4 – Діаграма варіантів використання Адміністратор

3.2.1 Опис варіантів використання

Прецедент «Вхід до системи»

Призначення: даний варіант використання надає можливість зареєстрованому користувачу увійти у систему для використання.

Основний потік подій: даний варіант використання починає виконуватися, коли зареєстрованому користувачу потрібно авторизуватися. Система пропонує ввести логін та пароль. Після того, як користувач ввів їх, система перевіряє правильність введених даних, і у випадку вдачі надає йому функціонал.

Альтернативний потік: якщо логін чи пароль невірні – система сповіщає про це користувача. Користувач може спробувати ще раз пройти авторизацію.

Передумова: перед початком виконання даного варіанта використання користувач повинен бути зареєстрований у системі.

Прецедент «Вихід з системи»

Призначення: даний варіант використання надає можливість користувачу вийти з системи.

Основний потік подій: даний варіант використання починає виконуватися, коли користувачу потрібно вийти з системи. Користувач натискає на значок профіля та у списку дій обирає «Вихід». Після того, як користувач вийшов, система відображає головну сторінку.

Передумова: перед початком виконання даного варіанта використання користувач повинен авторизуватися в системі.

Прецедент «Робота з замовленнями на переклад»

Призначення: даний варіант використання надає можливість користувачу взаємодіяти з замовленнями на переклад згідно з системною роллю.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає на кнопку «Замовлення» в особистому кабінеті. Система відображає список замовлень згідно з системною роллю.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід до системи.

Прецедент «Змінити статус»

Призначення: даний варіант використання надає можливість користувачу змінити статус замовлення.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи обирає замовлення та натискає кнопку «Змінити статус», система відображає інтерфейс зміни статуса. Користувач

обирає потрібний статус на натискає кнопку «Зберегти», система зберігає зміни та відображає список замовлень.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися у розділі «Замовлення».

Виняткова ситуація 1: користувач натиснув кнопку «Скасувати» – система відображає сторінку «Замовлення».

Прецедент «Переглянути замовлення»

Призначення: даний варіант використання надає можливість користувачу переглянути деталі замовлення.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи обирає замовлення та натискає кнопку «Переглянути замовлення», система відображає сторінку з детальним описом замовлення.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися у розділі «Замовлення».

Прецедент «Експорт файлу»

Призначення: даний варіант використання надає можливість користувачу експортувати файл, прикріплений до замовлення.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає кнопку «Експорт файлу», система завантажує файл, прикріплений до замовлення.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися на сторінці детального опису замовлення.

Прецедент «Імпорт файлу»

Призначення: даний варіант використання надає можливість користувачу додати файл до замовлення.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає кнопку «Додати файл», система додає файл до замовлення.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися на сторінці детального опису замовлення.

Прецедент «Надіслати переклад електронною поштою»

Призначення: даний варіант використання надає можливість користувачу надіслати переклад на електронну адресу замовника.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач системи натискає кнопку «Надіслати електронною поштою», система надсилає файл з перекладом на електронну адресу замовника.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися на сторінці детального опису замовлення.

Виняткова ситуація 1: користувач не додав файл до замовлення – система відображає відповідне повідомлення та пропонує користувачу додати файл.

Виняткова ситуація 2: клієнт електронної пошти не відповідає – система відображає відповідне повідомлення.

Прецедент «Редагувати замовлення»

Призначення: даний варіант використання надає можливість адміністратору редагувати деталі замовлення.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи обирає замовлення та натискає кнопку «Редагувати», система відображає інтерфейс редагування замовлення.

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Замовлення».

Виняткова ситуація 1: адміністратор не обрав замовлення – кнопка «Редагувати» неактивна.

Прецедент «Видалити замовлення»

Призначення: даний варіант використання надає можливість адміністратору видалити замовлення.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи обирає замовлення та натискає кнопку «Видалити», система видаляє замовлення.

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Замовлення».

Виняткова ситуація 1: адміністратор не обрав замовлення – кнопка «Видалити» неактивна.

Прецедент «Призначити замовлення перекладачу»

Призначення: даний варіант використання надає можливість адміністратору призначити замовлення перекладачу.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи обирає замовлення та натискає кнопку «Призначити перекладачу», система відображає інтерфейс вибору перекладача. Після вибору перекладача зі списку, адміністратор натискає кнопку «Зберегти», система зберігає зміни.

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Замовлення».

Виняткова ситуація 1: адміністратор не обрав замовлення – кнопка «Призначити замовлення перекладачу» неактивна.

Прецедент «Змінити перекладача»

Призначення: даний варіант використання надає можливість адміністратору змінити перекладача замовлення.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи обирає замовлення та натискає кнопку «Змінити перекладача», система відображає інтерфейс вибору перекладача. Після вибору перекладача зі списку, адміністратор натискає кнопку «Зберегти», система зберігає зміни.

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Замовлення».

Виняткова ситуація 1: адміністратор не обрав замовлення – кнопка «Змінити перекладача» неактивна.

Прецедент «Відмовитися від замовлення»

Призначення: даний варіант використання надає можливість перекладачу відмовитися від замовлення.

Основний потік подій: даний варіант використання починає виконуватися, коли перекладач обирає замовлення та натискає кнопку «Відмовитися», система надсилає повідомлення адміністратору та переводить замовлення в статус «Нове замовлення».

Передумова: перед початком виконання даного варіанта використання перекладач повинен знаходитися в розділі «Замовлення».

Виняткова ситуація 1: прекладач не обрав замовлення – кнопка «Відмовитися від замовлення» неактивна.

Прецедент «Редагувати дані профілю»

Призначення: даний варіант використання надає можливість перекладачу редагувати особисту інформацію профілю.

Основний потік подій: даний варіант використання починає виконуватися, перекладач натискає на значок профіля та у списку дій обирає «Редагувати». Система відображає інтерфейс редагування профілю.

Передумова: перед початком виконання даного варіанта використання користувач повинен авторизуватися в системі.

Прецедент «Робота з перекладачами»

Призначення: даний варіант використання надає можливість адміністратору взаємодіяти з профілями перекладачів.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає кнопку «Перекладачі», система відображає перелік перекладачів.

Передумова: перед початком виконання даного варіанта використання користувач повинен авторизуватися в системі з правами адміністратора.

Прецедент «Додати перекладача»

Призначення: даний варіант використання надає можливість адміністратору додавати нових перекладачів.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає кнопку «Додати», система відображає форму створення нового перекладача. Після заповнення форми

валідними даними, адміністратор натискає кнопку «Зберегти», система зберігає нового перекладача та відображає сторінку «Перекладачі».

Альтернативний потік: якщо введені дані невалідні, то система сповістить про це та запропонує адміністратору ввести дані знов.

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Перекладачі».

Виняткова ситуація 1: введено невалідні дані – система відобразить відповідне повідомлення. Адміністратор може змінити дані.

Виняткова ситуація 2: перекладач вже є в системі – система відобразить відповідне повідомлення. Адміністратор може змінити дані.

Виняткова ситуація 3: адміністратор натиснув кнопку «Скасувати» – система відобразить сторінку «Перекладачі».

Прецедент «Редагувати перекладача»

Призначення: даний варіант використання надає можливість адміністратору редагувати профіль перекладача.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи натискає кнопку «Редагувати», система відображає форму редагування перекладача. Після заповнення форми валідними даними, адміністратор натискає кнопку «Зберегти», система зберігає зміни та відображає сторінку «Перекладачі».

Альтернативний потік: якщо введені дані невалідні, то система сповістить про це та запропонує адміністратору ввести дані знов.

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Перекладачі».

Виняткова ситуація 1: введено невалідні дані – система відобразить відповідне повідомлення. Адміністратор може змінити дані.

Виняткова ситуація 2: змінені дані відповідають даним іншого перекладача – система відобразить відповідне повідомлення. Адміністратор може змінити дані.

Виняткова ситуація 3: адміністратор натиснув кнопку «Скасувати» – система відобразить сторінку «Перекладачі».

Прецедент «Видалити перекладача»

Призначення: даний варіант використання надає можливість адміністратору видалити профіль перекладача.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи обирає перекладача та натискає кнопку «Видалити», система видаляє викладача та відображає сторінку «Перекладачі».

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Перекладачі».

Виняткова ситуація 1: адміністратор не обрав перекладача – кнопка «Видалити» неактивна.

Прецедент «Переглянути дані перекладача»

Призначення: даний варіант використання надає можливість адміністратору переглянути профіль перекладача.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор системи обирає перекладача та натискає кнопку «Переглянути», система відображає сторінку, яка містить дані перекладача.

Передумова: перед початком виконання даного варіанта використання адміністратор повинен знаходитися в розділі «Перекладачі».

Виняткова ситуація 1: адміністратор не обрав перекладача – кнопка «Переглянути» неактивна.

Прецедент «Замовити переклад»

Призначення: даний варіант використання надає можливість клієнту замовити переклад.

Основний потік подій: даний варіант використання починає виконуватися, коли клієнт переходить на головну сторінку сайту та натискає кнопку «Замовити переклад». Система відображає форму створення замовлення. Після введення валідних даних та прикріплення файлу для

перекладу, система відображає інформацію про замовлення. Клієнт натискає кнопку «Оформити замовлення», система зберігає замовлення.

Передумова: перед початком виконання даного варіанта використання клієнт повинен знаходитися на головній сторінці сайту.

Виняткова ситуація 1: клієнт ввів невалідні дані – система відобразить відповідне повідомлення під невалідними полями. Клієнт може ввести дані знов.

Виняткова ситуація 2: перевищено розмір файлу для завантаження – система відобразить відповідне повідомлення. Клієнт може завантажити новий файл.

Виняткова ситуація 3: невірний тип завантаженого файлу – система відобразить відповідне повідомлення. Клієнт може завантажити новий файл.

Прецедент «Залишити відгук»

Призначення: даний варіант використання надає можливість клієнту залишити відгук.

Основний потік подій: даний варіант використання починає виконуватися, коли клієнт переходить на головну сторінку сайту та натискає кнопку «Залишити відгук». Система відображає форму створення відгука. Після введення валідних даних, клієнт натискає кнопку «Зберегти», система зберігає відгук.

Передумова: перед початком виконання даного варіанта використання клієнт повинен знаходитися на головній сторінці сайту.

Виняткова ситуація 1: клієнт ввів невалідні дані – система відобразить відповідне повідомлення під невалідними полями. Клієнт може ввести дані знов.

3.3 Діаграма діяльності

При моделюванні поведінки проектованої або аналізованої системи виникає необхідність не тільки подати процес зміни її станів, але і деталізувати

особливості алгоритмічної і логічної реалізації операцій, які виконуються системою. Традиційно для цієї мети використовувалися блок-схеми або структурні схеми алгоритмів. Кожна така схема акцентує увагу на послідовності виконання певних дій або елементарних операцій, які в сукупності призводять до отримання бажаного результату.

Для моделювання процесу виконання операцій в мові UML використовуються так звані діаграми діяльності. Кожен стан на діаграмі діяльності відповідає виконанню деякої елементарної операції, а перехід в наступний стан спрацьовує тільки при завершенні цієї операції в попередньому стані.

Діаграми діяльності відіграють важливу роль для розуміння потоків управління в системі, яка моделюється. Традиційні блок-схеми алгоритмів, які використовуються для цієї мети, мають серйозні обмеження в поданні паралельних процесів і їх синхронізації. Застосування доріжок і об’єктів відкриває додаткові можливості для наочного уявлення процесів, дозволяючи специфікувати діяльність підрозділів організації. Таким чином, ця діаграма будується для окремого класу, варіанти використання, окремої операції класу або цілої підсистеми.

Діяльністю (Activity) називається поведінка, реалізоване об’єктом, коли він знаходиться в даному стані. Діяльність – поведінка, яка може припинятися. Вона може виконуватися до свого завершення, якщо об’єкт знаходиться в даному стані, або може бути перервана переходом об’єкта в інший стан.

Наведемо діаграму діяльності, що описує модель поведінки варіанта використання «Замовити переклад». Діаграма представлена на рисунках 3.5 – 3.6.

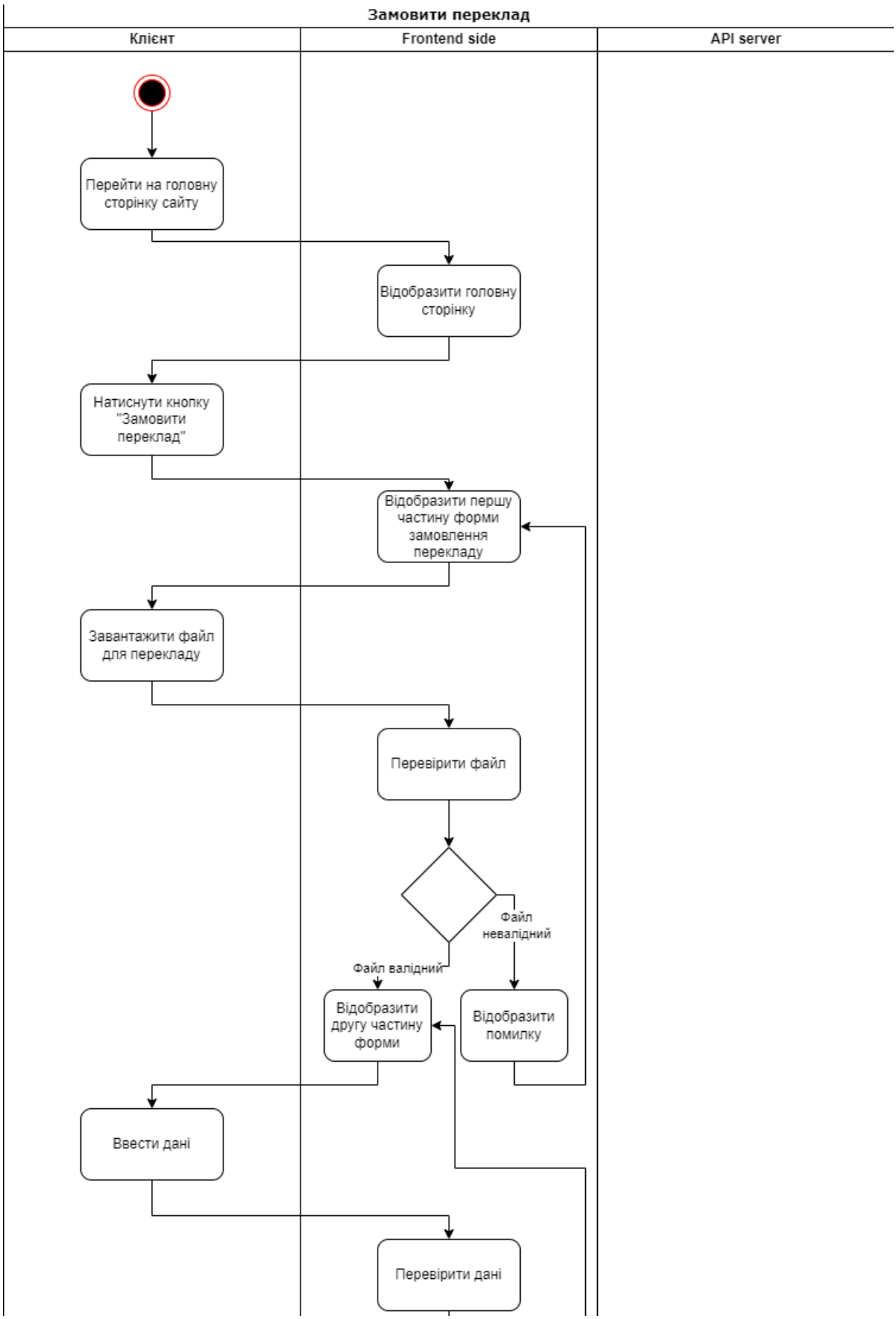


Рисунок 3.5 – Діаграма діяльності

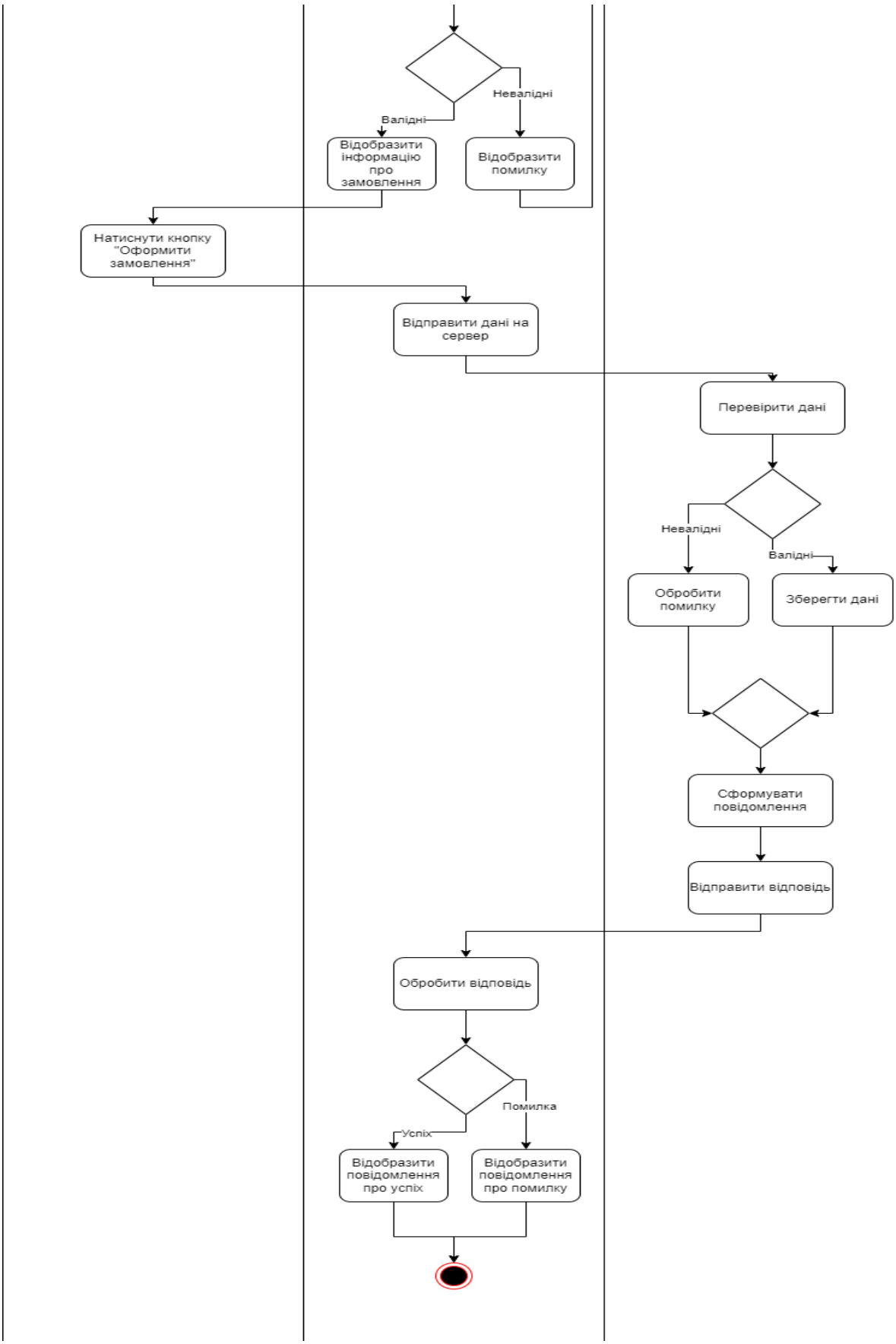


Рисунок 3.6 – Діаграма діяльності

3.4 Діаграма послідовності

Відображає взаємодії об’єктів впорядкованих за часом. Зокрема, такі діаграми відображають задіяні об’єкти та послідовність відправлених повідомлень.

Іншими словами, діаграма послідовностей відображає часові особливості передачі і прийому повідомлень об’єктами.

Діаграми послідовностей можна використовувати для уточнення діаграм прецедентів, більш детального опису логіки сценаріїв використання. Це відмінний засіб документування проекту з точки зору сценаріїв використання. Діаграми послідовностей зазвичай містять об’єкти, які взаємодіють у рамках сценарію, повідомлення, якими вони обмінюються, і які повертаються результати, які пов’язані з повідомленнями.

На діаграмі послідовності зображаються тільки ті об’єкти, які безпосередньо беруть участь у взаємодії.

Лінія життя об’єкта зображується пунктирною вертикальною лінією, асоційованою з єдиним об’єктом на діаграмі послідовності. Лінія життя служить для позначення періоду часу, протягом якого об’єкт існує в системі і, отже, може потенційно брати участь у всіх її взаємодіях. Якщо об’єкт існує в системі постійно, то і його лінія життя повинна продовжуватися по всій площині діаграми послідовності від самої верхньої її частини до самої нижньої.

У процесі функціонування об’єктно-орієнтованих систем одні об’єкти можуть перебувати в активному стані, безпосередньо виконуючи певні дії, або стані пасивного очікування повідомлень від інших об’єктів. Щоб явно виділити подібну активність об’єктів, в мові UML застосовується спеціальне поняття, що отримало назву фокуса управління . Фокус управління зображується у формі витягнутого вузького прямокутника, верхня сторона якого позначає початок отримання фокусу управління об’єкта, а його нижня сторона – закінчення фокусу управління. Прямокутник розташовується нижче

позначення відповідного об’єкта і може змінювати його лінію життя, якщо на всьому її протязі він є активним.

В UML кожна взаємодія описується сукупністю повідомлень, якими ті об’єкти, що беруть участь у ньому обмінюються між собою. Повідомлення є закінченим фрагментом інформації, який відправляється одним об’єктом іншому. Прийом повідомлення ініціює виконання певних дій, спрямованих на вирішення окремого завдання тим об’єктом, якому це повідомлення відправлено.

На рисунку 3.7 описана діаграма послідовності прецедента «Замовити переклад».

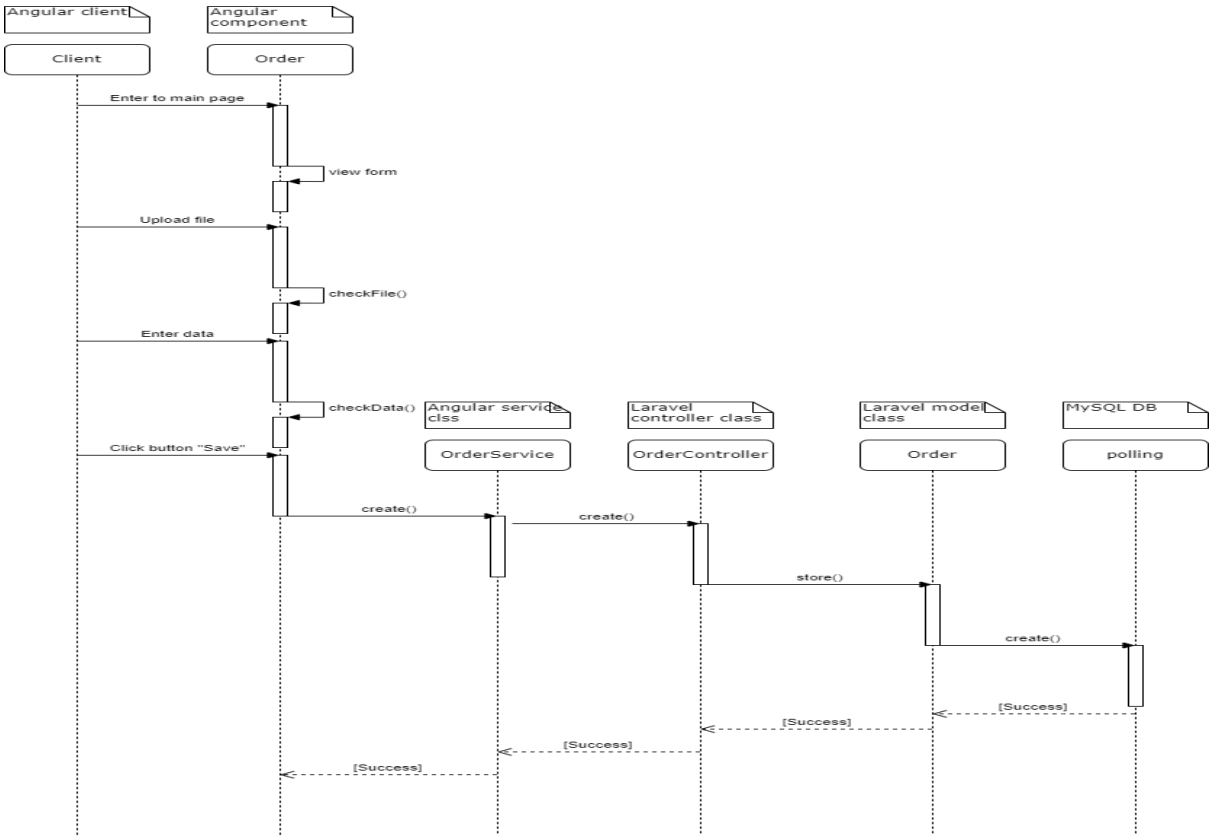


Рисунок 3.7 – Діаграма послідовності

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Опис інструментів розробки

Для реалізації були використані php-фреймворк Laravel та front-end фреймворк Angular.

RxJS – це бібліотека, що реалізує принципи реактивного програмування JavaScript. Заснована на об’єктах типу Observable, вона спрощує написання та контроль асинхронного та подвійного коду.

Angular Material – це бібліотека, що включає 30 сучасних компонентів, виготовлених згідно специфікації Google Material Design.

4.2 Процес створення Angular-компонента

Є два шляхи створення компоненту в Angular:

– консольна команда;

– вручну.

Процес створення компонента за допомогою консольної команди:

– відкрити консоль (термінал) у корні проекту;

– ввести команду «ng g c %path_to_component/%%component_name% --skip-tests».

Розглянемо аргументи команди:

– g – скорочення від generate, генерує та/або змінює файли на основі схеми;

– c – скорочення від component, схема для створення;

– %path_to_component/% – шлях, по якому буде згенеровано компонент, якщо відсутній, то створюється у директорії src/app;

– %component_name% – назва компоненту;

– skip-tests – опціональний аргумент, не створює тестовий файл для компоненту.

Після виконання команди, система створить новий компонент з базовими файлами шаблону, стилів та оброблювача. Також система зареєструє новий компонент у масиві declarations, файла app.module.ts.

Процес створення компонента вручну:

– відкрити проект у зручному редакторі коду (наприклад WebStorm);

– створити теку з ім’ям компоненту в теці app, це буде контейнер для файлів компонента;

– створюємо у теці компоненту файли app.%component_name%.html|.scss|.ts;

– у файлі app.%component_name%.ts визначаємо декоратор @Component, у якому підключаємо файли шаблону та стилів;

– експортуємо класс компонента, для можливості його використання у інших частинах проекту;

– імпортуємо компонент у файл app.module.ts та реєструємо його у масиві declarations.

Візуалізація процесу та приклад коду компонента наведено у Додатку А.

4.3 Процес створення Angular-сервісу

Є два шляхи створення сервісу в Angular:

– консольна команда;

– вручну.

Процес створення сервісу за допомогою консольної команди:

– відкрити консоль (термінал) у корні проекту;

– ввести команду «ng g s %path_to_service/%%service_name% --skip-tests».

Розглянемо аргументи команди:

– s – скорочення від service, схема для створення;

– %path_to_service/% – шлях, по якому буде згенеровано сервіс, якщо відсутній, то створюється у директорії src/app;

– %service_name% – назва сервісу;

– skip-tests – опціональний аргумент, не створює тестовий файл для сервісу.

Після виконання команди, система створить новий сервіс.

Процес створення сервісу вручну:

– відкрити проект у зручному редакторі коду (наприклад WebStorm);

– створити теку з назвою «services» в теці app, це буде контейнер для файлів сервісів;

– створюємо у теці «services» файл app.%service_name%.ts;

– у файлі app.%service_name%.ts визначаємо декоратор @Injectable;

– експортуємо класс сервісу, для можливості його використання у інших частинах проекту.

Візуалізація процесу та приклад коду сервісу наведено у Додатку Б.

4.4 Процес створення Laravel-моделі

Є два шляхи створення моделі в Laravel:

– консольна команда;

– вручну.

Процес створення моделі за допомогою консольної команди:

– відкрити консоль (термінал) у корні проекту;

– ввести команду «php artisan make:model %path_to_model/%%model_name% [params]».

Розглянемо аргументи команди:

– make:model – створює новий клас Eloquent моделі;

– %path_to_model/% – шлях, по якому буде згенеровано модель, якщо відсутній, то створюється у директорії app/Models;

– %model_name% – назва моделі;

– params – можно вказати додаткові аргументи для автоматичного створення контролера, міграції, тощо.

Після виконання команди, система створить нову модель.

Процес створення моделі вручну:

– відкрити проект у зручному редакторі коду (наприклад PhpStorm);

– створюємо у теці «app/Models» клас %model_name%.php;

– у файлі %model_name%.php вказуємо, що вона буде розширяти базову модель «Illuminate\Database\Eloquent\Model».

Візуалізація процесу та приклад коду моделі наведено у Додатку В.

4.5 Процес створення Laravel-контролеру

Є два шляхи створення контролеру в Laravel:

– консольна команда;

– вручну.

Процес створення контролеру за допомогою консольної команди:

– відкрити консоль (термінал) у корні проекту;

– ввести команду «php artisan make:controller %path_to_controller/%%controller_name% [params]».

Розглянемо аргументи команди:

– make:controller – створює новий клас контролера;

– %path_to_controller/% – шлях, по якому буде згенеровано контроллер, якщо відсутній, то створюється у директорії app/Http/Controllers;

– %controller_name% – назва контролеру;

– params – можно вказати додаткові аргументи для автоматичного ресурсів та прив’язки до моделі.

Після виконання команди, система створить новий контролер.

Процес створення контролеру вручну:

– відкрити проект у зручному редакторі коду (наприклад PhpStorm);

– створюємо у теці «app/Http/Controllers» клас %controller_name%.php;

– у файлі %controller_name%.php вказуємо, що він буде розширяти базовий контролер «App\Http\Controllers\Controller».

Візуалізація процесу та приклад коду моделі наведено у Додатку Г.

4.6 Основні класи системи

Так як основним прецедентом системи є «Замовити переклад», то основними класами системи будуть ті, що надають можливість створювати, змінювати, видаляти та переглядати дані.

Головними класами зі сторони Angular є:

– OrderService – відповідає за дії пов’язані з Laravel API-сервером, а саме містить запити на створення, видалення, зміну та відображення даних про замовлення;

– OrderComponent – обробляє та відображає дані, отримані від методів OrderService. Містить методи для управління відображенням прив’язаного до компоненту шаблона. Отримує дані від шаблону та передає їх у сервіс.

Головними класами зі сторони Laravel є:

– Order – модель, що пов’язана з таблицею «orders». Містить список полів для запису до таблиці, їх типи, зв’язки з іншими таблицями;

– OrderController – групує логіку обробки запитів пов’язаних з моделлю «Order».

Нижче приведено приклади функцій контролеру та сервісу.

show(): Observable<any> {

```
    return this.http.get(this.apiUrl + '/order'); }
  public function show() {
    $orders = Order::with(['statuses', 'users'])->get();
    if($orders === []) {
        return response()->json(['message' => 'No matches found'], 200);
     }
   return response()->json(['orders' => $orders, 'message' => 'Success'],
  200);
  }
```

Детально ознайомитися з кодом проекту можна у додатку Д.

4.7 Ієрархія даних проекту

4.7.1 Angular

Каталог src/app містить основний код програми.

Файл src/index.html – основна html-сторінка програми.

Файл src/main.ts відповідає за запуск програми.

Файл src/tsconfig.app.json відповідає за компіляцію TypeScript.

Компоненти знаходяться у каталозі src/app/components.

Сервіси знаходяться у каталозі src/app/services.

Кастомні інтерсептори, валідатори та пайпи знаходяться у папках src/app/interceptors, src/app/validators, src/app/pipes.

У папці src/app/core знаходяться інтерфейси та moc-дані.

4.7.2 Laravel

Головний index.php лежить у каталозі public. В цьому каталозі також знаходяться зображення, стилі, тощо.

Основний код програми лежить у каталозі app.

Каталог bootstrap містить файл app.php, котрий завантажує фреймворк. Також містить каталог cache, який містить файли згенеровані фреймворком (наприклад, файли кешу маршрутів та служб).

Каталог config містить файли конфігурації програми.

Каталог database містить міграції до БД, фабрики моделей та наповнювачі.

Каталог routes містить визначення маршрутів. Для моєї системи використовується api.php, котрий містить маршрути для API-запитів.

Каталог storage містить журнали (логи), скомпіловані шаблони Blade, файли сесій, кеша та інші файли, створені фреймворком.

Каталог vendor містить Composer-залежності.

Каталог Http містить контролери, посередники та запити форм. Практично вся логіка обробки запитів, що надходять програму, буде розміщена в цьому каталозі.

Каталог Models містить усі класи моделей Eloquent.

Файл .env містить конфігурацію оточення (наприклад, підключення до БД).

4.8 Тестування проекту

4.8.1 Unit-тест

Модульне тестування, або юніт-тестування – процес у програмуванні, що дозволяє перевірити на коректність окремі модулі вихідного коду програми.

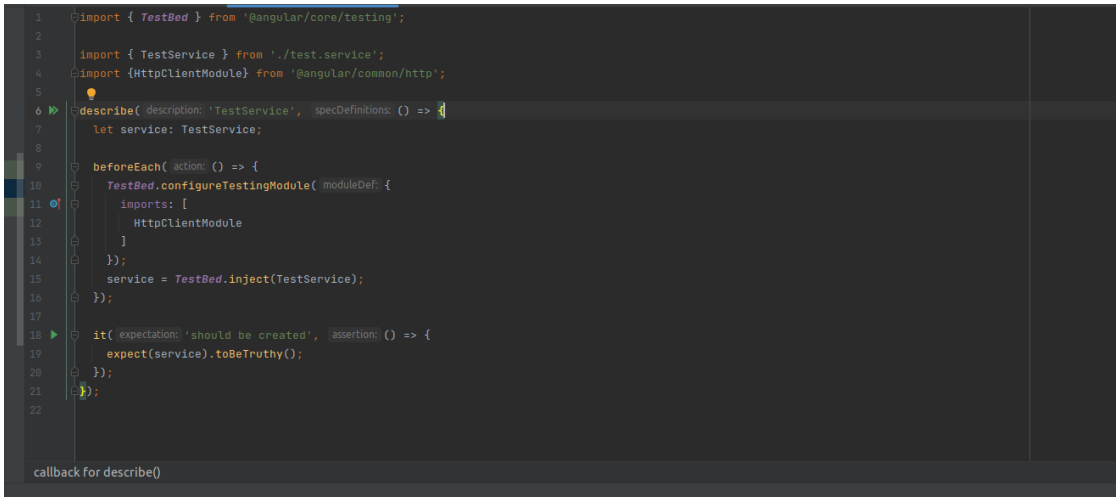
Ідея полягає в тому, щоб писати тести для кожної нетривіальної функції чи методу. Це дозволяє досить швидко перевірити, чи не привела чергова зміна коду до регресії, тобто до появи помилок у вже відтестованих місцях програми, а також полегшує виявлення та усунення таких помилок. Приклади такого тестування наведено на рисунку 4.1.

```
beforeEach( action: () => {
  // @ts-ignore
  component = new SigninComponent( router: null,  authService: null,  token: null,  authState: null);
});

it( expectation: 'should create form with 2 controls',  assertion: () => {
  expect(component.form.contains('email')).toBe( expected: true);
  expect(component.form.contains('password')).toBe( expected: true);
});

it( expectation: 'should mark email as invalid if empty value',  assertion: () => {
  const control = component.form.get('email');
  control?.setValue( value: '');
  expect(control?.valid).toBeFalse();
});

it( expectation: 'should mark password as invalid if empty value',  assertion: () => {
  const control = component.form.get('password');
  control?.setValue( value: '');
  expect(control?.valid).toBeFalse();
});

it( expectation: 'should mark email as valid if valid value',  assertion: () => {
  const control = component.form.get('email');
  control?.setValue( value: 'test@gmail.com');
  expect(control?.valid).toBeTruthy();
});

it( expectation: 'should mark password as valid if valid value',  assertion: () => {
  const control = component.form.get('password');
  control?.setValue( value: '123456');
  expect(control?.valid).toBeTruthy();
});

it( expectation: 'should mark password as invalid if length < 6',  assertion: () => {
  const control = component.form.get('password');
  control?.setValue( value: '12345');
  expect(control?.valid).toBeFalse();
```

Рисунок 4.1 – Тестування SigninComponent

4.8.2 Integration-тест

Інтеграційне тестування проводиться для тестування модулів/компонентів, коли вони інтегровані, щоб перевірити, чи працюють вони належним чином. На рисунку 4.2 зображено тестування на створення сервісу.



```
import { TestBed } from '@angular/core/testing';

import { TestService } from './test.service';
import {HttpClientModule} from '@angular/common/http';

describe( description: 'TestService',  specDefinitions: () => {
  let service: TestService;

  beforeEach( action: () => {
    TestBed.configureTestingModule( moduleDef: {
      imports: [
        HttpClientModule
      ]
    });
    service = TestBed.inject(TestService);
  });

  it( expectation: 'should be created',  assertion: () => {
    expect(service).toBeTruthy();
  });
});

callback for describe()
```

Рисунок 4.2 – Тестування створення сервісу

4.8.3 Чек-лист верстки

Чек-лист – це документ, що описує, що має бути протестоване. При цьому чек-лист може бути абсолютно різного рівня деталізації. Він дозволяє не забувати про важливі тести, фіксувати результати своєї роботи і відслідковувати статистику про статус програмного продукту.

	Google Chrome	FF	Safari	Opera	Notes
Наличие элементов страниц					
Проверить корректное отображение страниц согласно файлам дизайна	Passed	Passed	Passed	Passed	
Проверить наличие логотипа	Passed	Passed	Passed	Passed	
Проверить наличие главного меню	Passed	Passed	Passed	Passed	
Проверить наличие подвала сайта	Passed	Passed	Passed	Passed	
Отображение элементов					
Проверить корректное отображение шрифта текста	Passed	Passed	Passed	Passed	
Проверить корректное отображение кнопок, блоков меню и т.д.	Passed	Passed	Passed	Passed	
Проверить отображение цветовой гаммы всех элементов	Passed	Passed	Passed	Passed	
Проверить корректное размещение баннеров	Passed	Passed	Passed	Passed	
Активные элементы					
Проверить изменение курсора после наведения на активные элементы	Passed	Passed	Passed	Passed	
Проверить реагирование активных элементов на наведение	Passed	Passed	Passed	Passed	
Содержание страниц					
Проверить орфографию/грамматику контента сайта	Passed	Passed	Passed	Passed	
Новые пункты					
Проверить корректное расположение элементов header	Passed	Passed	Passed	Passed	
Проверить корректное отображение элементов body	Passed	Passed	Passed	Passed	
Проверить корректное расположение элементов footer	Passed	Passed	Passed	Passed	
Проверить страницу на повторяющиеся элементы	Passed	Passed	Passed	Passed	
Проверить расположение элементов страницы относительно друг друга	Passed	Passed	Passed	Passed	
Проверить корректное отображения элементов страницы при смене способа отображения	Passed	Passed	Passed	Passed	

Рисунок 4.3 – Чек-лист верстки

4.9 Керівництво користувача

4.9.1 Рівень підготовки користувача

Користувач сайту повинен володіти певною кваліфікацією.

Навички користувача для роботи з ПК, та навички роботи з web-браузером.

Знайомство з Керівництвом користувача.

4.9.2 Підготовка до роботи

Запуск системи.

Доступ до сайту здійснюється через мережу Інтернет за допомогою звичайного web-браузера. Адреса сайту в мережі Інтернет: http://polling.loc. Для коректної роботи клієнтської частини повинен використовуватися браузер Google Chrome, Mozilla Firefox, Opera, Safari.

У системі розрізняються наступні групи користувачів:

Клієнт – приватна особа, що має можливість робити замовлення та залишати відгук.

Перекладач – приватна особа, авторизований на сайті (далі Користувач), що має можливість редагування профілю та має доступ до функцій користувача у особистому кабінеті.

Адміністратор системи (далі Адміністратор) відповідає за підтримку та конфігурування системи.

4.9.3 Вхід та завантаження файлу

При вході на сайт, Ви потрапляєте на головну сторінку (рис. 4.4). Для того щоб відразу перейти до замовлення перекладу, потрібно натиснути кнопку «Замовити переклад».

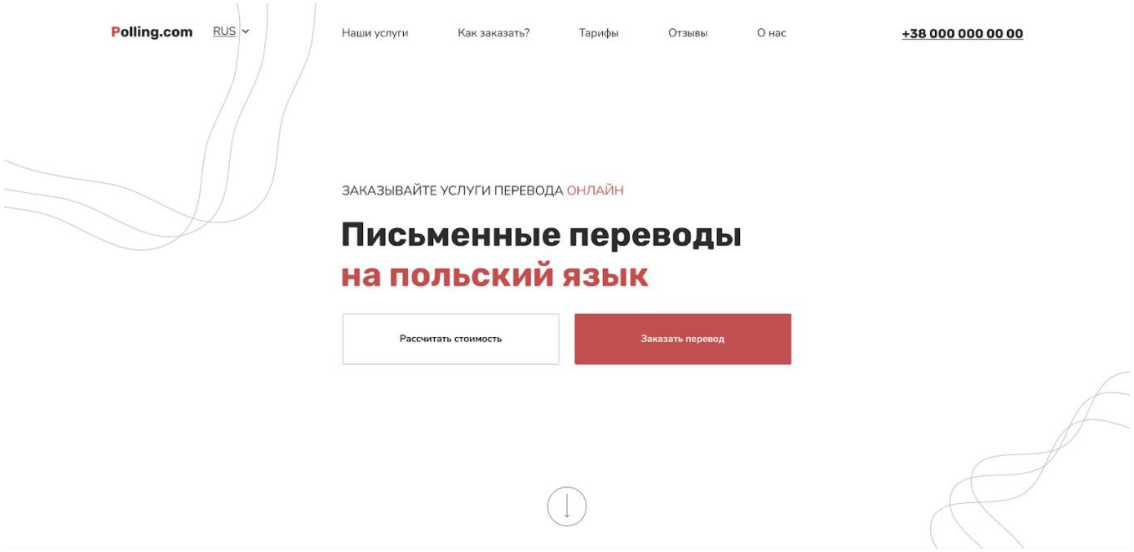


Рисунок 4.4 – Головна сторінка сайту

Система відобразить форму для замовлення переказу та запропонує завантажити документ для перекладу (рис. 4.5).

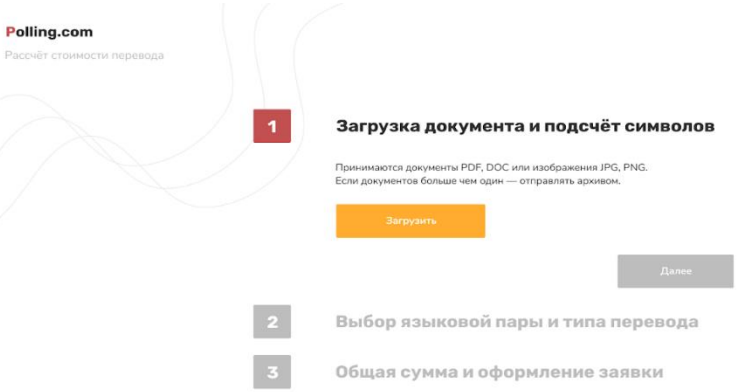


Рисунок 4.5 – Завантаження документа

У разі завантаження невалідного файлу, система сповістить про це у вигляді повідомлення біля кнопки «Завантажити» (рис. 4.6). Якщо файл валідний, то система розблокує кнопку «Далі» (рис. 4.7).

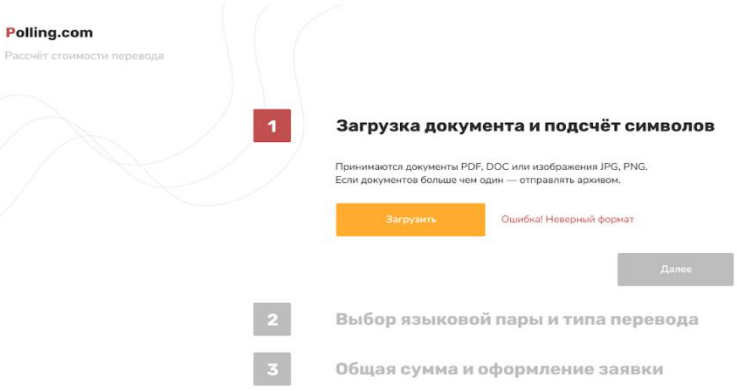


Рисунок 4.6 – Невалідний файл

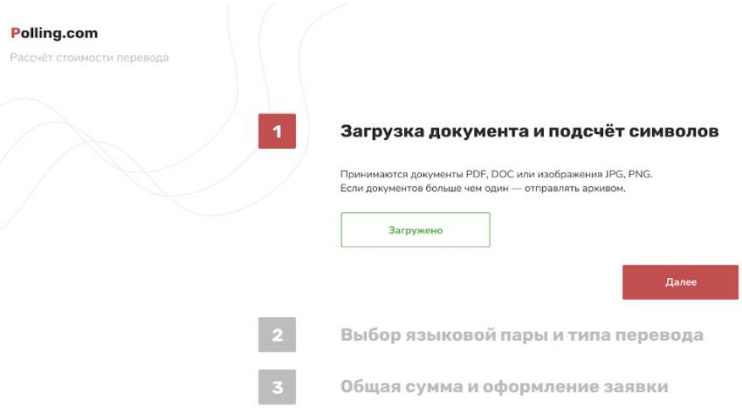


Рисунок 4.7 – Валідний файл

Після натискання кнопки «Далі», система відобразить другу частину форми (рис. 4.8).

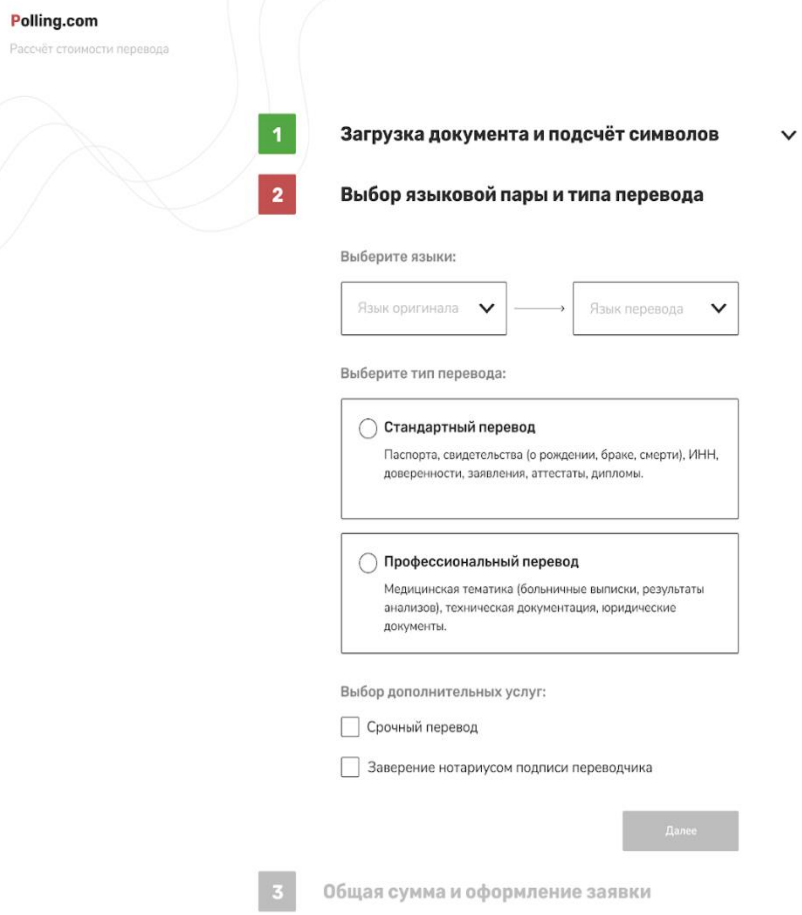


Рисунок 4.8 – Форма обрання типу та мови перекладу

4.9.4 Обрання типу та мови перекладу

Після відображення другої частини форми, потрібно заповнити поля типу та мови перекладу, а також обрати додаткові функції, за потребності. Якщо форма валідна, то система розблокує кнопку «Далі» (рис. 4.9).

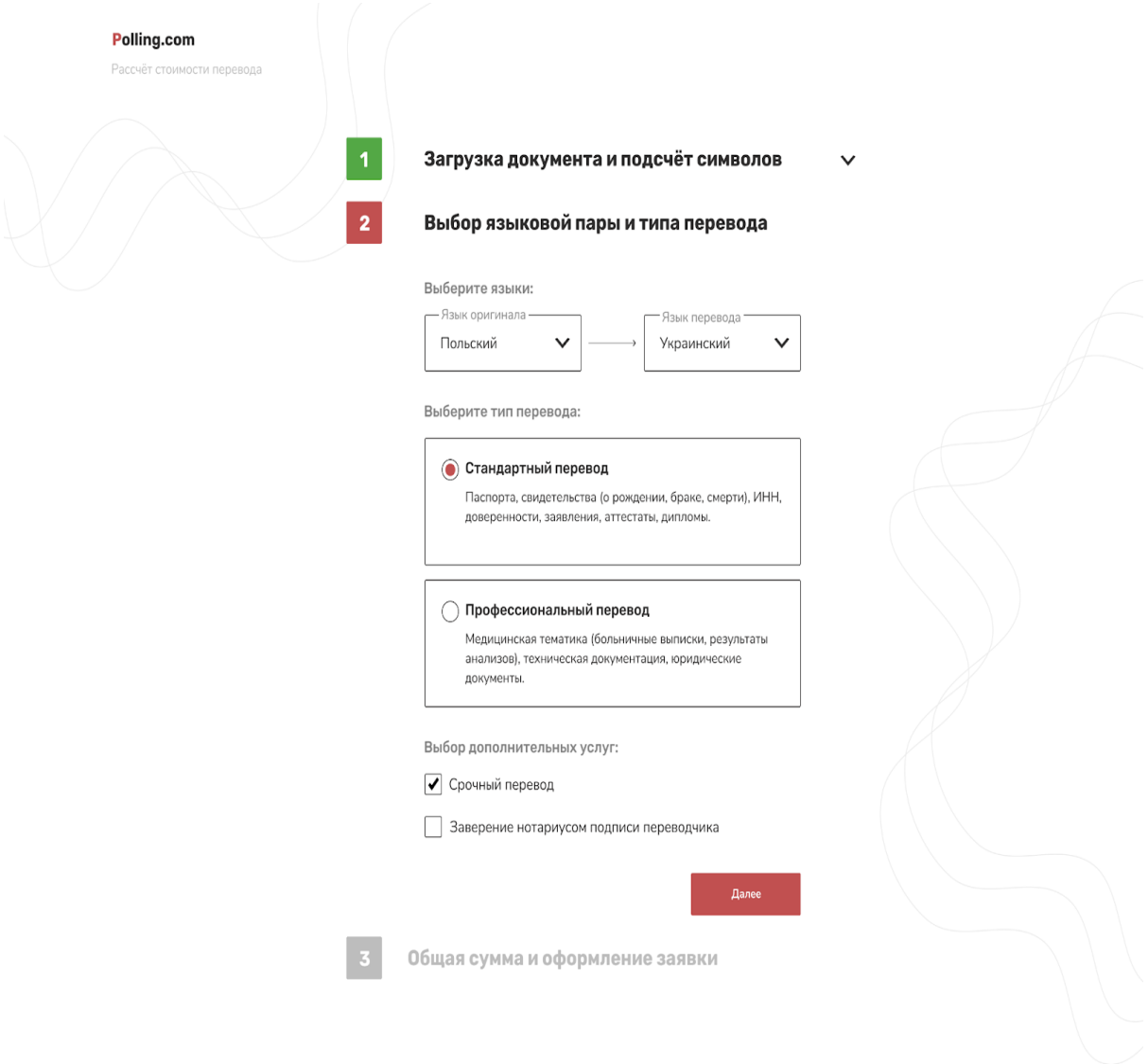


Рисунок 4.9 – Валідна форма

Якщо елементи форми невалідні, то система сповістить про це, відобразивши помилки під невалідними полями форми. Ще одним захисним

механізмом, що сигналізує про невалідність форми – заблокована кнопка «Далі» (рис. 4.10).

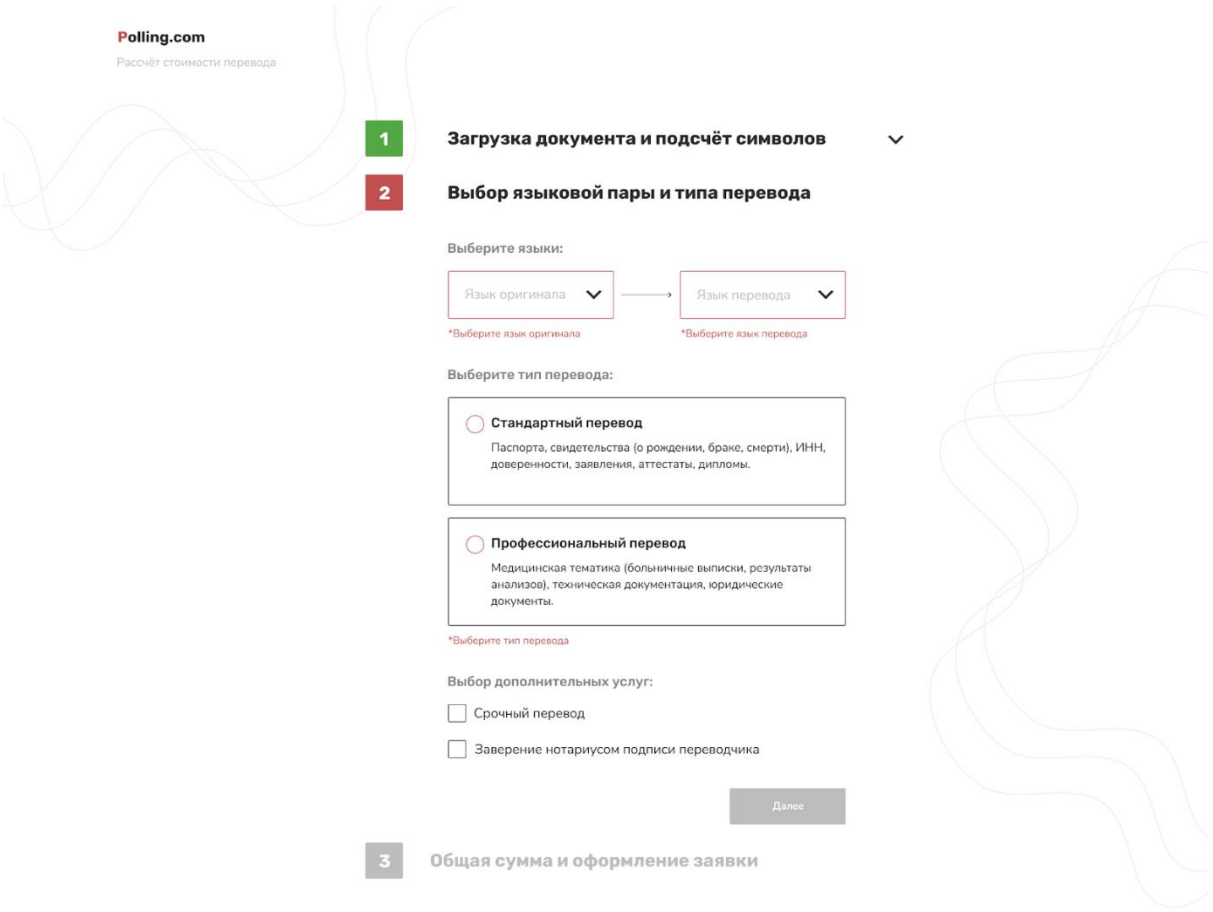


Рисунок 4.10 – Невалідна форма

Після натискання кнопки «Далі», система відобразить другу частину форми (рис. 4.11).

4.9.6 Перевірка та оформлення замовлення

Після відображення системою третьої частини форми, Вам потрібно перевірити, чи всі дані введені вірно. Якщо був завантажений текстовий файл, то система відразу порахує кількість знаків та відобразить мінімальну суму замовлення. Сума може змінитися після розгляду замовлення перекладачем, про що Ви будете сповіщені на Ваш email. Якщо завантажений файл не є

документом, то система відобразить повідомлення, що для уточнення фінальної суми з Вами зв’яжеться перекладач. Також при необхідності можна вказати кінцевий строк виконання замовлення. Після замовлення перекладу, система відобразить повідомлення про успішне подання заявки (рис. 4.11-4.15).

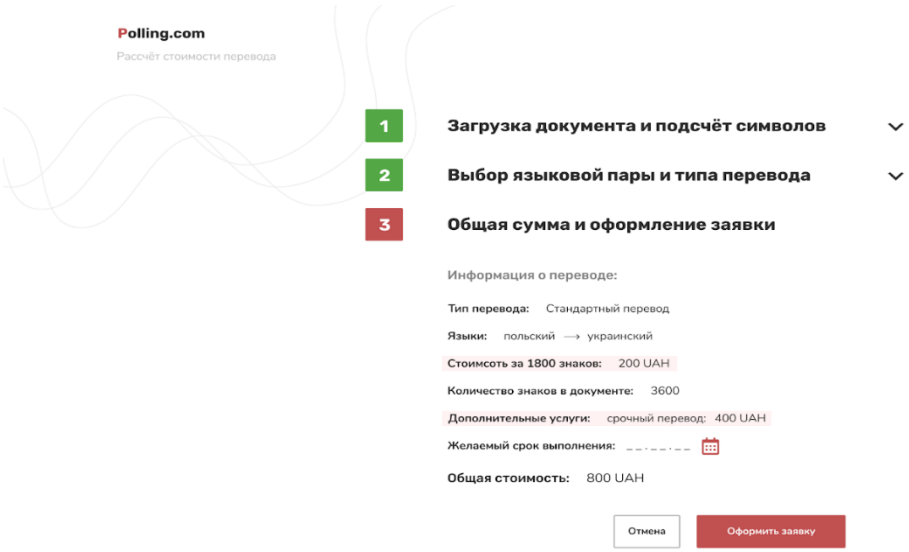


Рисунок 4.11 – Інформація про замовлення

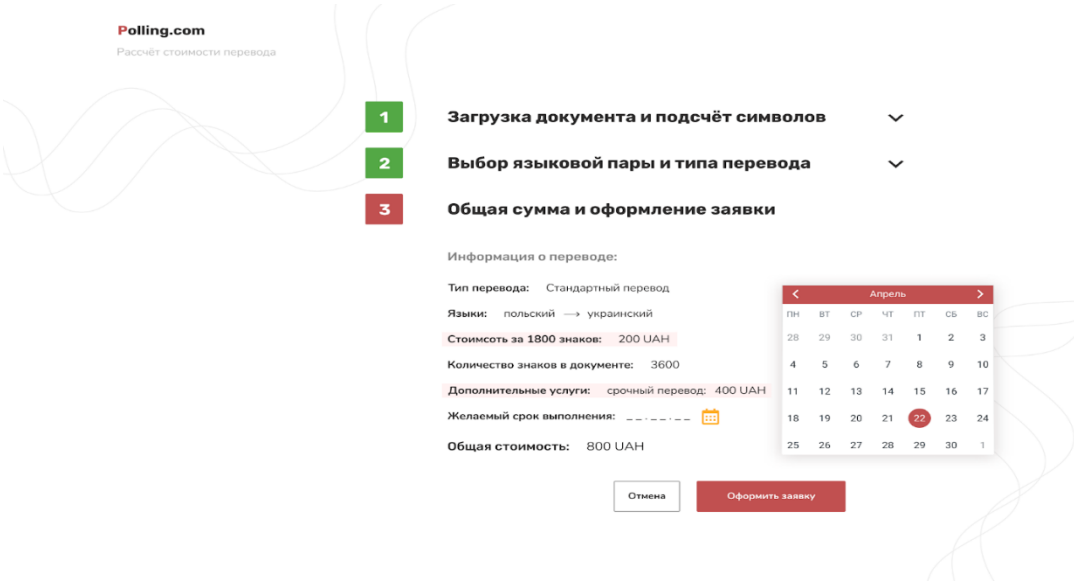


Рисунок 4.12 – Вибір кінцевого строку замовлення

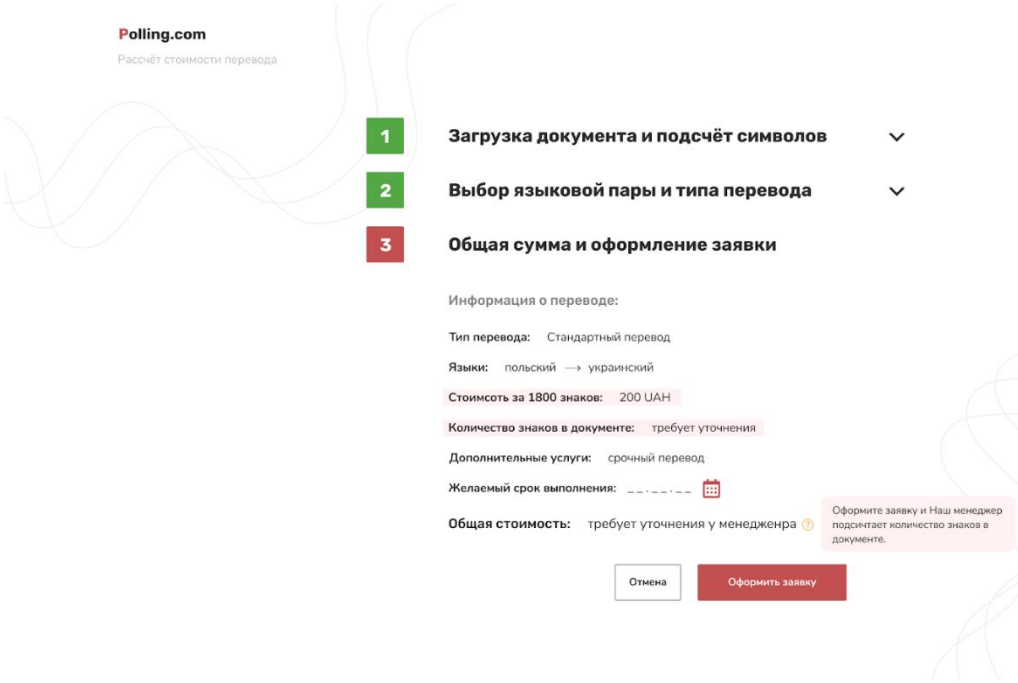


Рисунок 4.13 – Відображення суми у разі завантаження зображення або архіву

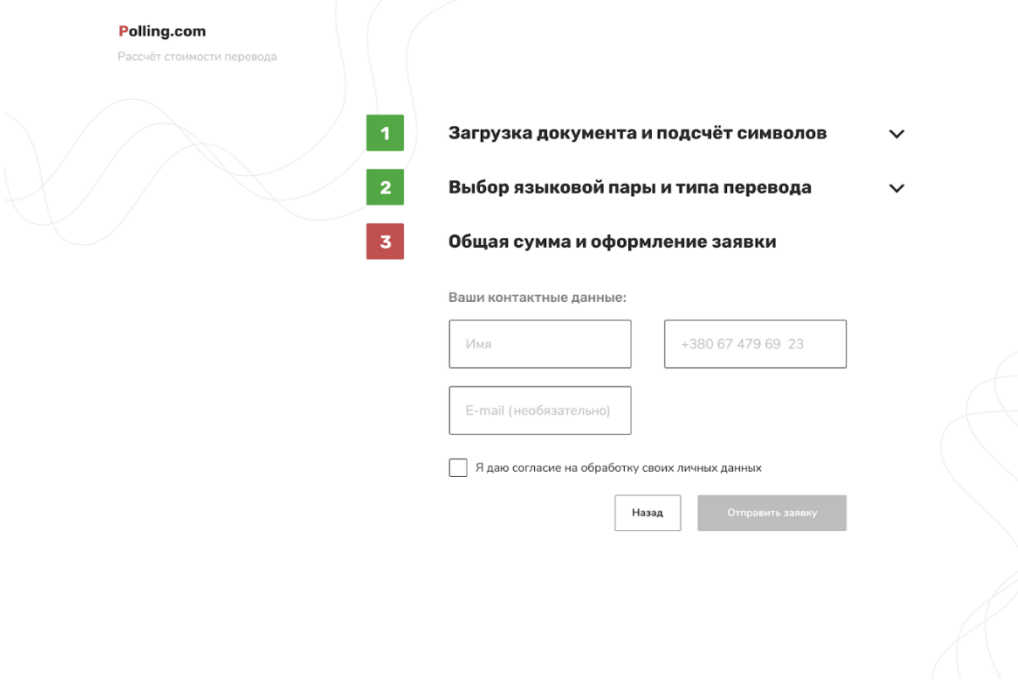


Рисунок 4.14 – Ввод контактних даних

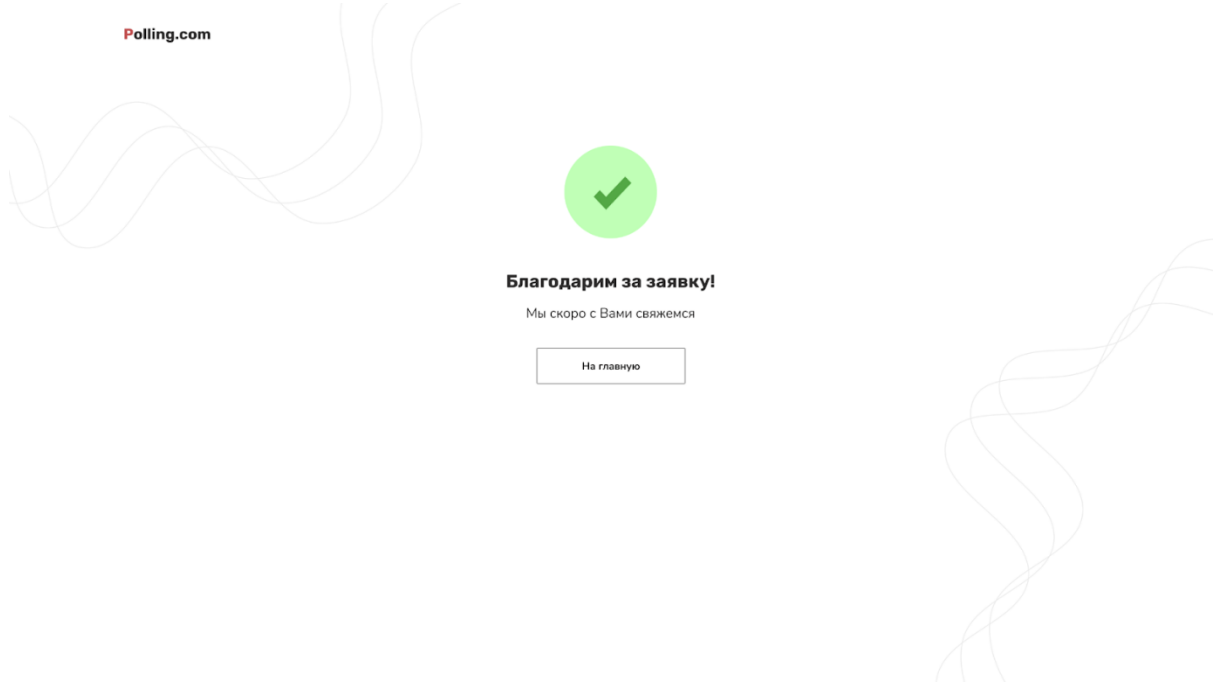


Рисунок 4.15 – Повідомлення про успішне подання заявки

4.9.7 Залишити відгук

Система надає можливість залишити відгук, для цього потрібно на головній сторінці сайту перейти до секції «Відгуки» (рис. 4.16) та натиснути кнопку «Залишити відгук». Система відобразить форму залишення відгуку (рис. 4.17).

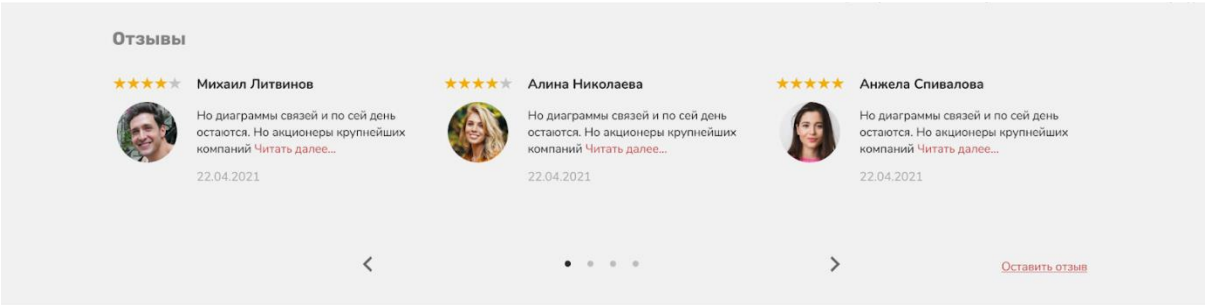


Рисунок 4.16 – Секція «Відгуки»

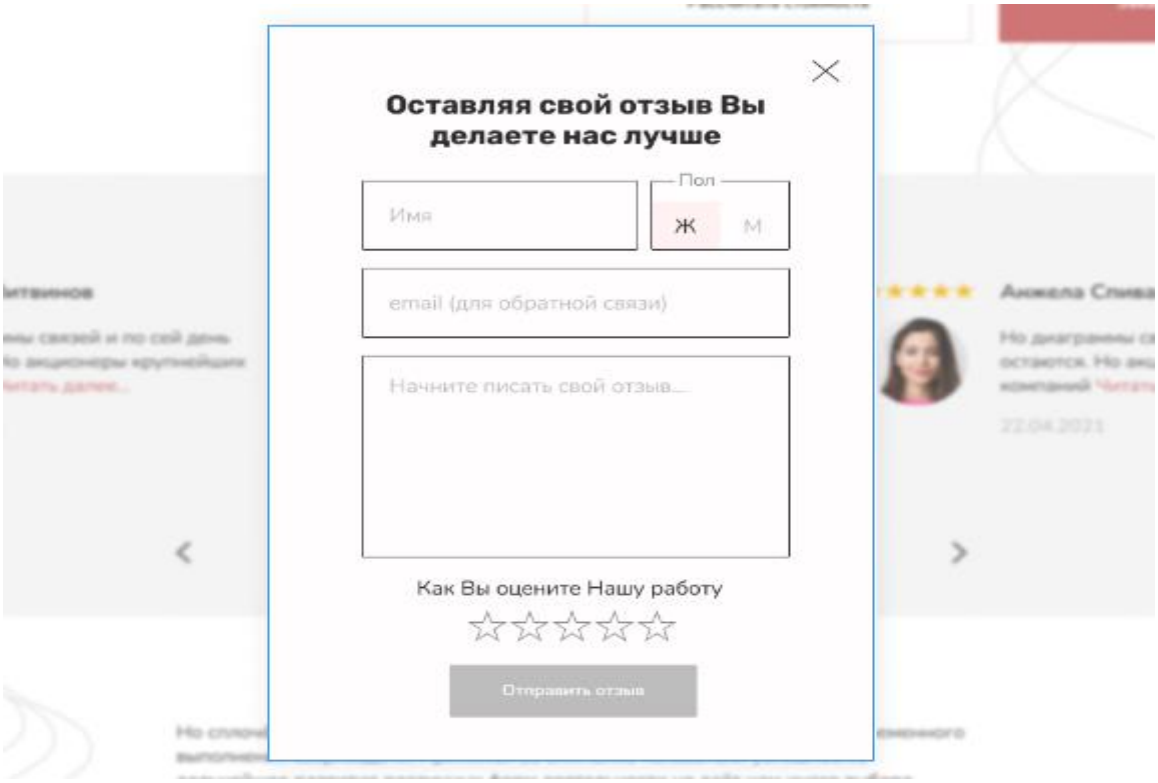


Рисунок 4.17 – Форма відгуку

Після введення валідних даних та натискання кнопки «Відправити відгук», система відобразить повідомлення про успіх (рис. 4.18).

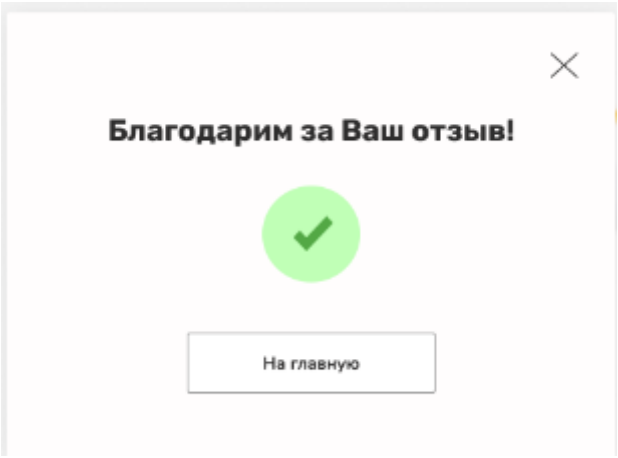


Рисунок 4.18 – Повідомлення про успішне відправлення відгуку

ВИСНОВКИ

В результаті роботи було написано технічне завдання на розробку системи замовлення перекладів. Для створення цієї системи були обрані популярні фреймворки Angular та Laravel, за її широкі можливості у сфері створення web-систем.

У відповідності з метою кваліфікаційної роботи була розроблена система замовлення перекладів із застосуванням наступних технологій:

- Laravel для реалізації back end API;
- Angular для реалізації front end частини, а також відправки запитів до серверу.

У відповідності з поставленими задачами були виконані наступні етапи створення системи:

- сформовані вимоги до системи (функціональні та нефункціональні (інтерфейс, кросбраузерність, безпека, продуктивність)). Також проведено огляд предметної області та інструментів розробки;
- спроектована та побудована структура системи (побудовані діаграми прецедентів, діяльності та послідовності; надано детальний опис прецедентів);
- реалізована система замовлення перекладів (наведена інструкція по створенню компонентів системи, надано керівництво користувача та структура проекту);
- протестована робота системи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Angular Developer Documentation. URL: https://angular.io/docs/ (дата звернення: 11.09.2022).

2. JSON Web Token Authentication for Laravel & Lumen. URL: https://jwt-auth.readthedocs.io/en/develop/ (дата звернення: 13.09.2022).

3. Laravel 8 Developer Documentation. URL: https://laravel.com/docs/8.x/ (дата звернення: 22.09.2022).

4. Laravel MongoDB. URL: https://github.com/jenssegers/laravel-mongodb/ (дата звернення: 22.09.2022).

5. Matt Stauffer. Laravel: Up & Running: A Framework for Building Modern PHP Apps / Matt Stauffer. : O'Reilly Media, 2019. 544 p.

6. MySQL Documentation. URL: https://dev.mysql.com/doc/ (дата звернення: 23.09.2022).

7. Yakov Fain ,Anton Moiseev. Angular Development with TypeScript / Yakov Fain. New York : Manning Publications, 2019. 560 p.

8. Адам Фримен. Angular для профессионалов / Адам Фримен. Санкт-Петербург : Питер Пресс, 2020. 800 с.

9. Владимир Дронов. Laravel 8. Быстрая разработка веб-сайтов на PHP / Владимир Дронов. Санкт-Петербург : БХВ-Петербург, 2021. 688 с.

10. Сергей Трофимов. CASE-технологии. Практическая работа в Rational Rose / Сергей Трофимов. Москва :Бином-Пресс, 2002. 288 с.

11. Хансен Г. Базы данных: разработка и управление / Г. Хансен. М. :ЗАО Издательство «БИНОМ», 1999. 704 с.

12. Эванс В.М. Предметно-ориентированное проектирование / Эванс В.М. М. :СИНТ, 2010. 448 с.

ДОДАТОК А

Angular-компонент

А.1 Приклад створення компонента

ng g c component/test –skip-tests

```
CREATE src/app/components/test/test.component.scss (0 bytes)
CREATE src/app/components/test/test.component.html (19 bytes)
CREATE src/app/components/test/test.component.ts (268 bytes)
UPDATE src/app/app.module.ts (2668 bytes)
```

Рисунок А.1 – Створення компонента за допомогою консольної команди

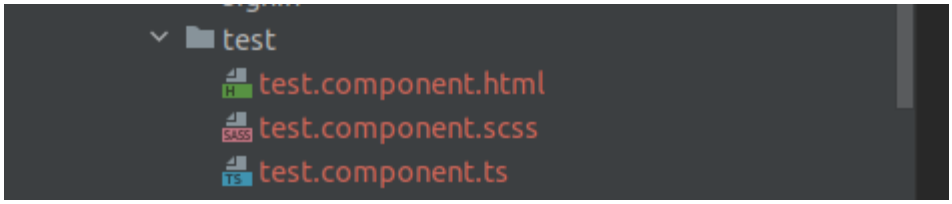


Рисунок А.2 – Демонстрація створеного компоненту

test.component.ts

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-test',
  templateUrl: './test.component.html',
  styleUrls: ['./test.component.scss']
})
export class TestComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {
  }

}
```

Рисунок А.3 – Підключення файлів шаблону та стилів

А.2 Приклад компоненту

```
import {AfterViewInit, Component, OnInit, ViewChild} from '@angular/core';
import {OrderService} from '../../servises/order.service';
import {Order} from '../../core/models/order';
import {MatTableDataSource} from '@angular/material/table';
import {MatSort} from '@angular/material/sort';
import {MatPaginator} from '@angular/material/paginator';
import {FormControl} from '@angular/forms';
import {DatePipe} from '@angular/common';

@Component({
 selector: 'app- order ',
 templateUrl: './ order.component.html',
 styleUrls: ['./ order.component.scss']
})
export class OrderComponent implements OnInit, AfterViewInit {

 @ViewChild(MatSort) sort!: MatSort;
 @ViewChild(MatPaginator) paginator!: MatPaginator;

 public orders = new MatTableDataSource<Order>();
 public displayedColumns = ['user', 'value', 'date', 'details', 'update', 'delete'];

 isShowAddForm = false;
 isShowEditForm = false;
 pipe!: DatePipe;

 userFilter = new FormControl('');
 valueFilter = new FormControl('');
 fromDate = new FormControl();
 toDate = new FormControl();

 filterValues = {
  user: '',
  value: '',
  fromDate: Date,
  toDate: Date,
```

```
};

editData!: any;

constructor(
  public orderService: OrderService
) {
  this.pipe = new DatePipe('en');

  this. orderService.show().subscribe((data: any) => {
    this.orders.data = data.orders.sort((a: any, b: any) => {
      return (new Date(b.created_at) as any) – (new Date(a.created_at) as any);
    }) as Order[];
  });

  this.orders.filterPredicate = (data, filter) => {
    const searchTerms = JSON.parse(filter);
    if (searchTerms.fromDate && searchTerms.toDate) {
      return data?.categories?.name.trim().toLowerCase().indexOf(searchTerms.user) !== -1
        && data.value.toString().trim().toLowerCase().indexOf(searchTerms.value) !== -1
        && new Date(new Date(data.created_at).setHours(0, 0, 0, 0)) >= new Date(searchTerms.fromDate)
        && new Date(new Date(data.created_at).setHours(0, 0, 0, 0)) <= new Date(searchTerms.toDate);
    } else {
      return data?.categories?.name.trim().toLowerCase().indexOf(searchTerms.user) !== -1
        && data.value.toString().trim().toLowerCase().indexOf(searchTerms.value) !== -1;
    }
  };
}

ngOnInit(): void {
  this.userFilter.valueChanges.subscribe(
    user => {
      this.filterValues.user = user;
      this.orders.filter = JSON.stringify(this.filterValues);
  });
  this.valueFilter.valueChanges.subscribe(
    value => {
      this.filterValues.value = value;
```

```
      this.orders.filter = JSON.stringify(this.filterValues);
    }
  );
  this.fromDate.valueChanges.subscribe(
    from => {
      this.filterValues.fromDate = from;
      this.orders.filter = JSON.stringify(this.filterValues);
    }
  );
  this.toDate.valueChanges.subscribe(
    to => {
      this.filterValues.toDate = to;
      this.orders.filter = JSON.stringify(this.filterValues);
    }
  );
}

 ngAfterViewInit(): void {
  this.orders.sort = this.sort;
  this.orders.paginator = this.paginator;
 }

 openAddForm(): void {
  this.isShowAddForm = true;
 }

 closeAddForm(event: any): void {
  this.isShowAddForm = event;
 }

openEditForm(element: any): void {
 this.editData = element;
 this.isShowEditForm = true;
}

closeEditForm(event: any): void {
 this.isShowEditForm = event;
}
```

```
addOrder(order: Order): void {
  this.orders.data.unshift(order);
  this.orders._updateChangeSubscription();
}

delete(id: number): void {
  this.orderService.delete(id).subscribe(
    result => {
      this.orders.data = this.orders.data.filter(item => item.id !== id);
      this.orders._updateChangeSubscription();
    },
    error => {
      console.log(error);
    }
  );
}
```

ДОДАТОК Б

Angular-сервіс

Б.1 Приклад створення сервісу

ng g s services/test –skip-tests

Рисунок Б.1 – Створення сервісу за допомогою консольної команди

Рисунок Б.2 – Демонстрація створеного сервісу

Б.2 Приклад сервісу

```
import { Injectable } from '@angular/core';
import {HttpClient, HttpHeaders} from '@angular/common/http';
import {Observable} from 'rxjs';
import {Order} from '../core/models/order';

@Injectable({
 providedIn: 'root'
})
export class OrderService {
 apiUrl = 'http://127.0.0.1:8000/api';
 options: any;

 constructor(private http: HttpClient) {
   this.options = {
     headers: new HttpHeaders({
```

```
      Accept: ‘application/json’,
      ’Content-Type’: ‘application/json’
    })
  };
}

show (): Observable<any> {
  return this.http.get(
    this.apiUrl + ‘/orders’,
    this.options
  );
}

create(order: Order): Observable<any> {
  return this.http.post(
    this.apiUrl + ‘/orders’,
    order,
    this.options
  );
}

update(id: number, body: any): Observable<any> {
  return this.http.patch(
    this.apiUrl + ‘/orders/’ + id,
    body,
    this.options
  );
}

delete(id: number): Observable<any> {
  return this.http.delete(
    this.apiUrl + ‘/orders/’ + id,
    this.options
  );
 }
}
```

ДОДАТОК В

Laravel-модель

В.1 Приклад створення моделі

php artisan make:model Test

Рисунок В.1 – Створення моделі за допомогою консольної команди

В.2 Приклад моделі

```
<?php

namespace App\Models;

use App\Traits\Date;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Order extends Model
{
  use HasFactory, Date;

  protected $table = ‘orders’;

  const CREATED_AT = ‘created_at’;
  const UPDATED_AT = ‘updated_at’;

  protected $fillable = [
     ’value’,
     ’status’,
     ‘language_start’,
```

```
        'language_finish',
        'type',
        'email',
        'user_id',
    ];


    public function users() {
        return $this->belongsTo(User::class,'user_id');
    }
}
```

ДОДАТОК Г

Laravel-контролер

Г.1 Приклад створення контролера

php artisan make:controller TestController

Рисунок Г.1 – Створення контролера за допомогою консольної команди

Г.2 Приклад контролера

```
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Models\Order;
use Illuminate\Support\Facades\Validator;
use Illuminate\Http\Request;

class OrderController extends Controller
{
  /**
   * @return \Illuminate\Http\JsonResponse
   */
  public function index() {
     $orders = Order::with(['users'])->get();
     return response()->json(['orders' => $orders, 'message' => 'Success'], 200);
  }

  /**
   * @param Request $request
```

```
     * @return \Illuminate\Http\JsonResponse
     */
    public function showByUserId($id) {
        $orders = Order::with(['users'])->get()->where('user_id', $id);
        if($orders === []) {
            return response()->json(['message' => 'No matches found'], 200);
        }

        return response()->json(['orders' => $orders, 'message' => 'Success'], 200);
    }

    /**
     * @param Request $request
     * @param $id
     * @return \Illuminate\Http\JsonResponse
     */
    public function update(Request $request, $id) {
        $data = $request->all();

        $order = Order::where('id', intval($id))->update($data,  ['upsert' => true]);

        return response()->json(['order' => $order, 'message' => 'Retrieved successfully'], 200);
    }

    /**
     * @param $id
     * @return \Illuminate\Http\JsonResponse
     */
    public function destroy($id) {
        Order::where('id', intval($id))->delete();
        return response()->json(['message' => 'Success'], 200);
    }
}
```

ДОДАТОК Д

Посилання на Git

https://bitbucket.org/s_var_og/polling/src/main/