ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**ВЕБ-СЕРВІС АСИСТЕНТ ДЛЯ СТУДЕНТІВ ТА УЧНІВ “ASSIST EASY” МОВОЮ C#**»

Виконав: студент 4 курсу, групи ПД-43
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)

Кирпичов О.П.
(прізвище та ініціали)

Керівник Поперешняк С.В.
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти -«Бакалавр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Негоденко О.В.
"____" _____2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТА

КИРПИЧОВА ОЛЕКСАНДРА ПЕТРОВИЧА

(прізвище, ім'я, по батькові)

1. Тема роботи: «Веб-сервіс асистент для студентів та учнів "Assist Easy" мовою C#»

Керівник роботи: Поперешняк С.В. к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «18» лютого 2022 року №__.
2. Строк подання студентом роботи «03» червня 2022 року
3. Вхідні дані до роботи
Методи обробки та зберігання даних;
Інструменти з розробки програмного забезпечення: SDK .NET 5, IDE JetBrains Rider 2021.3, IDE VS Code, DBeaver, Docker, Git, Azure DevOps, npm CLI
Науково-технічна література з питань, пов'язаних з розробкою програмного рішення на основі мікросервісів;
Технічна документація до прикладного програмного інтерфейсу ботів Telegram;

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).

4.1 Аналіз предметної області
4.2 Вимоги та оцінка якості системи
4.3 Проектування та реалізація веб-сервісу
4.4 Тестування системи

5. Перелік демонстраційного матеріалу (назва основних слайдів)
1. Мета, об'єкт та предмет дослідження
2. Актуальність роботи
3. Аналоги
4. Порівняння з аналогами
5. Технічне завдання
6. Програмні засоби реалізації
7. Інструменти використані для реалізації
8. Архітектура сервісу
9. Механізм автентифікації
10. Схема відправлення повідомлень
11. Заплановані сповіщення на основі Job-ів
12. Апробація результатів дослідження
13. Висновки
14. Перспективи подальших досліджень та розвитку даної роботи

6. Дата видачі завдання «11» квітня 2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	01.03.22	Виконано
2	Аналіз та дослідження існуючих аналогів	02.03.22-09.03.22	Виконано
3	Проектування системи	10.03.22-17.03.22	Виконано
4	Створення та тестування програмного рішення	18.03.22-10.04.22	Виконано
5	Підготовка розділу 1	11.04.22-13.04.22	Виконано
6	Підготовка розділу 2	14.04.22-17.04.22	Виконано
7	Підготовка розділу 3	18.04.22-26.04.22	Виконано
8	Підготовка розділу 4	26.04.22-27.04.22	
9	Вступ, висновки, реферат	28.04.22-30.04.22	Виконано
10	Розробка обов‘язкових демонстраційних матеріалів	01.05.22-05.05.22	Виконано
11	Попередній захист роботи та перевірка на плагіат	25.05.22	
12	Здача роботи	03.06.22	

Студент __ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 92 с., 118 рис., 15 табл., 2 дод., 16 джерел.

ВЕБ-СЕРВІС, ВЕБ ДОДАТОК, АСИСТЕНТ, МІКРОСЕРВІСИ, ASSIST EASY, GOOGLE CLASSROOM, TRELLO, ASP.NET CORE, WEB API, API, СЕРВІС ДЛЯ СТУДЕНТІВ ТА УЧНІВ

Об'єкт дослідження – здійснення навчання за допомогою інформаційних технологій.

Предмет дослідження – веб-сервіс асистент для студентів та учнів на основі мікросервісів.

Мета роботи – спрощення та прискорення процесу навчання для студентів та учнів за допомогою веб-сервісу та сучасних прикладних програмних інтерфейсів.

Методи дослідження – методи передачі, зберігання та обробки інформації, уніфікований процес створення програмного забезпечення, емпіричні методи.

У роботі проведено аналіз найближчих за функціоналом існуючих додатків, таких як Google Classroom та Moodle. Основними недоліками цих продуктів є відсутність можливості обирання декількох сучасних засобів для отримання інформації та сповіщень, відсутність можливості створення та адміністрування користувацьких динамічних сторінок, автоматичного розкладу онлайн занять та нативного механізму для формування рейтингу складності й статистики для завдань.

Науковою новизною даної роботи є відсутність абсолютних аналогів та реалізація функціоналу відсутнього в близьких за галуззю використання продуктах. Assist Easy дозволяє користувачам створювати та адмініструвати групи, що містять в собі предмети, домашні завдання, відстежуванні користувачем

завдання, вкладення у виді файлів, розклад онлайн занять та підтримку динамічну сторінку групи, де користувачі можуть самі побудувати сторінку та наповнити її необхідними розділами та додати вкладення у виді файлів та зображень. На відміну від аналогів підтримуються повідомлення засобами Telegram та підтримка вебхуків, а для завдань збирається статистика складності та загального прогресу серед групи.

Веб-сервіс складається з п’яти мікросервісів, а core додаток побудовано на основі RESTful API. Усі сервіси розроблено за допомогою фреймворку ASP.NET Core мовою програмування C# версії 9.0 (.NET 5). В якості сховищ даних було використано объектно-реляційну базу даних PostgreSQL та документно орієнтовану NoSQL базу даних MongoDB. Для збереження користувацьких зображень та файлів завдань було використано хмарне рішення від Microsoft під назвою Azure Blob Storage. Для надійної реалізації функцій відправлення повідомлень засобами email та telegram, а також відправлення користувацьких вебхуків використано брокер повідомлень RabbitMQ. Для забезпечення функції нотифікування та формування розкладу онлайн занять було реалізовано “polling” сповіщень та нагадувань за допомогою бібліотеки для створення Job-ів Quartz.Net. В якості провайдерів для сповіщень обрано email та telegram, для доставки повідомлень було використано бібліотеки MailKit та Telegram.Bot. Додаток побудовано за принципами SOLID із використанням Dependency Injection та модульної архітектури поділеної на шари.

Отже, розроблено та описано веб-сервіс, завданням якого є спрощення та підвищення швидкості процесу навчання для студентів та учнів.

Даний додаток може бути використано як для персональних цілей під час самоосвіти так і в сфері загальної та вищої освіти.

Галузь використання – освітній процес.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SDK – software development kit

API – Application Programming Interface

JSON – Java Script Object Notation

ORM – Object-Relation Mapping

БД – База Даних

СУБД - Система Управління Базами Даних

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

RPC – Remote procedure call

MVC – Model View Controller

ASP – Active Server Pages

SQL – Structured query language

AMQP – Advanced Message Queuing Protocol

SAS – Shared Access Signature

LINQ – Language-Integrated Query

IDE – Integrated Development Environment

JS – JavaScript

МБ - Мегабайт

ВСТУП

У сучасному світі інформаційних технологій, де вирує величезна кількість інформації яку потрібно впорядковувати, зберігати та слідкувати за нею, стає достатньо складно швидко та своєчасно керувати цим неспинним потоком даних. А особливо гостро це питання встає перед учнями й студентами, які практично живуть у цьому вирії знань. Ще більш складним процес навчання роблять певні глобальні на локальні зміни в житті. Одним з таких прикладів є пандемія.

Пандемія COVID-19 призвела до переведення значної кількості навчальних закладів на дистанційне навчання, але учбові заклади та учні не були готові до такої різкої зміни в навчальному процесі. Через неготовність до таких подій багато закладів не мали уніфікованого плану дій щодо організації дистанційних занять, а на студентів звалилося набагато більше матеріалу для опрацювання та завдань у порівнянні з очним навчанням. Через це стало достатньо складно тримати в голові весь список навчальних активностей, які потрібно зробити - відстежувати зміни, дедлайни, матеріали та вимоги до завдань, а також слідкувати за розкладом усіх онлайн занять, не загублюючи при цьому посилання на платформи, поширювати нову інформацію між студентами групи або класу, а також ділитися інформацією, порадами та питаннями з предметів між студентами.

Для вирішення вищезгаданих проблем потрібно мати комплексне рішення на основі інформаційних систем, які зможуть забезпечити швидку, своєчасну та зручну обробку даних, потрібних студентам та учням. Зможуть постійно тримати користувача в курсі останніх оновлень щодо навчального процесу, а також забезпечать можливість легкого адміністрування ресурсів групи. Також важливим аспектом навчання є збір та аналіз корисної персональної та групової статистики з метою подальшого аналізу та покращення процесу виконання завдань.

Серед сервісів подібних до вищеописаного можна виділити рішення від IT-гіганту компанії Google під назвою Google Classroom та платформу з відкритим

кодом Moodle. Але ці продукти мають певні недоліки та є більш орієнтованими на викладачів, через що існуючі рішення не покривають повний спектр проблем учнів та студентів. Тому призначенням веб-сервісу, що розробляється є створення зручного асистенту для навчання, який зможе задовільнити проблеми користувачів.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Важливою частиною розробки програмного забезпечення є аналіз предметної області. Оскільки предметною областю сервісу, що розроблюється, є створення асистенту з освітнього процесу для студентів та учнів, потрібно знайти та проаналізувати переваги й недоліки схожих за функціоналом та галуззю використання програмних продуктів.

1.1. Google Classroom

Рішення від компанії Google під назвою Google Classroom – це система управління контентом для шкіл, яка спрямована на спрощення створення, розповсюдження та оцінювання завдань. Сервіс був анонсований 6 травня 2014, а реліз сервісу відбувся 12 серпня 2014 року.

Google Classroom тісно пов'язаний з іншими сервісами компаніями, такими як Google Drive – де зберігаються завдання, Google Docs – де можна відкрити та редагувати матеріал, Sheets and Slides – для перегляду прещентацій, Gmail для спілкування та сповіщень та Google Calendar для розкладу. Здобувачі освіти можуть бути запрошені до класу через спеціальний код, або бути автоматично додані за допомогою шкільного сайту. Кожен клас виокристовує окрему папку на Google диску для відповідного користувача, куди зберігається робота учня, яку оцінює викладач. Classroom також має мобільні додатки, доступні на iOS та Android. Викладач може відстежувати прогрес кожного студента, оцінювати його роботи та залишати коментарі.

Рисунок 1. - Сторінка додатку Google Classroom

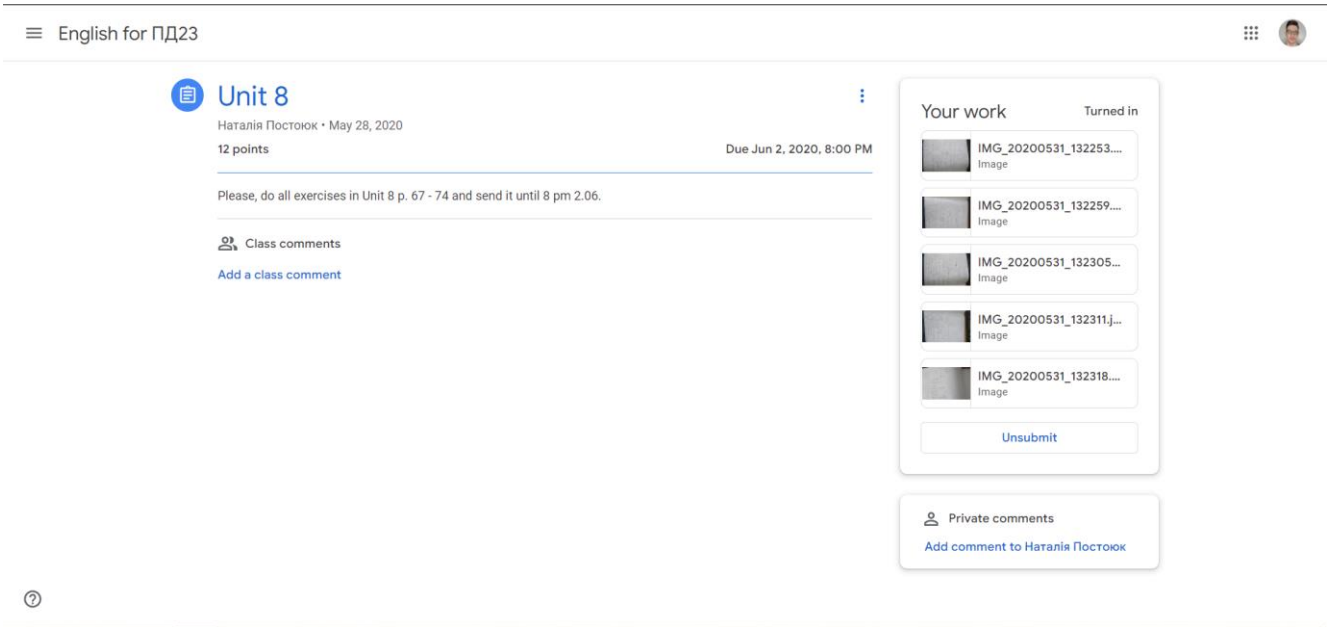


Рисунок 2. - Скріншот домашнього завдання в Google Classroom

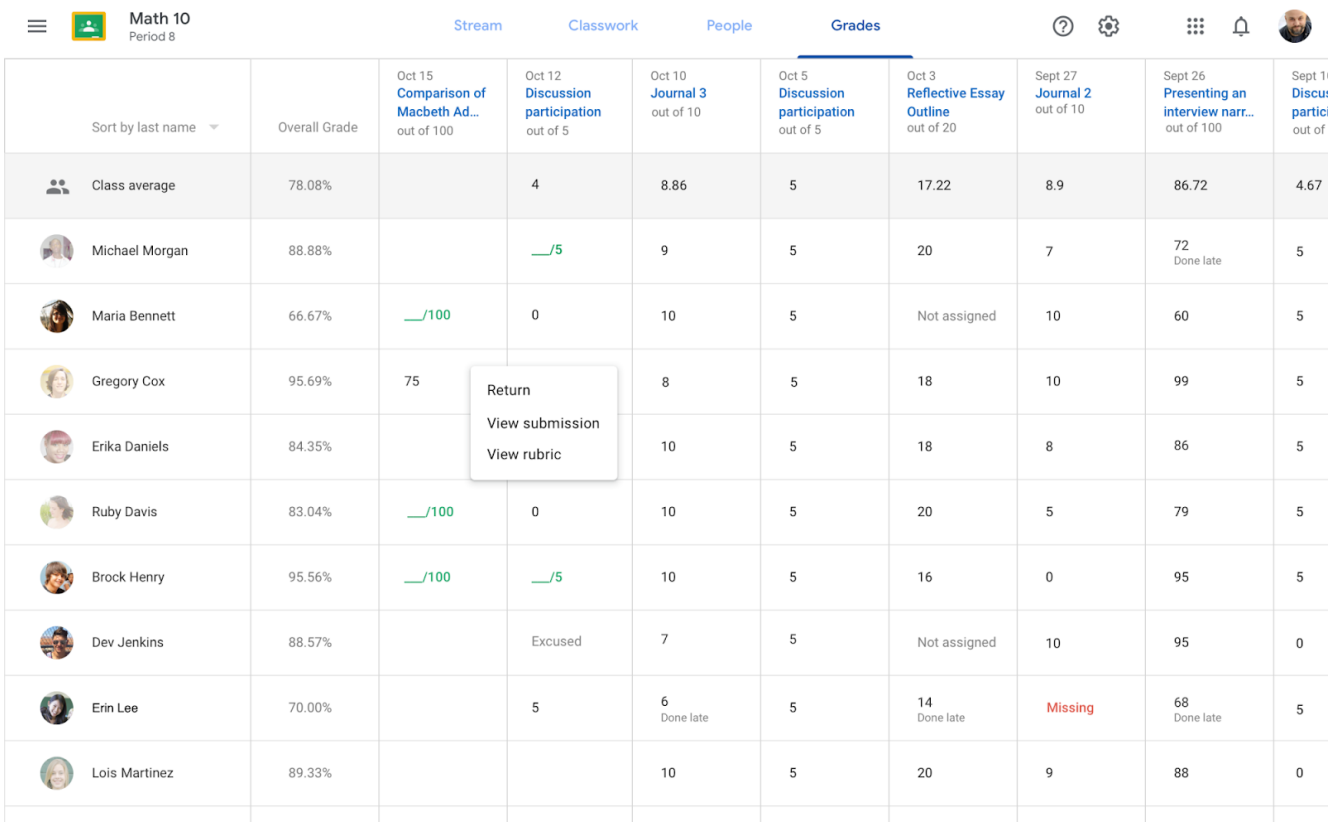


Рисунок 3. - Панель оцінювання учнів Google Classroom

Переваги Google Classroom:

- Інтеграція з іншими сервісами Google
- Відкритий API
- Можливість інтеграції із сайтами школи
- Підтримка мобільного додатку
- Система сповіщень за допомогою Gmail
- Наявність панелі керування для викладача
- Можливість створення завдань та прикріплення файлів вкладень
- Можливість залишати персональні та публічні коментарі до завдань
- Можливість оцінки завдань та перегляд оцінок учнем
- Система керування ролями користувачів

Недоліки Google Classroom:

- Закритий початковий код
- Орієнтованість на викладача

- Відсутність статистики складності завдань та загального прогресу групи
- Відсутність можливості отримання сповіщень через месенджер
- Відсутність підтримки користувацьких вебхуків
- Відсутність можливості створення динамічної головної сторінки класу
- Відсутність підтримки розкладу онлайн занять
- Відсутність користувацького TODO листу

1.2. Moodle

Рішення під назвою Moodle або Modular Object-Oriented Dynamic Learning Environment є модульним об'єктно-орієнтованим динамічним навчальним середовищем та навчальною платформою, призначеною для об'єднання педагогів, адміністраторів і студентів в одну надійну, інтегровану та безпечну екосистему для створення навчального середовища.

Головним розробником системи є Мартін Дугіамас з Австралії. Проєкт є відкритим (Open Source) і в ньому беруть участь інші розробники зі всього світу. Moodle написано на мові програмування PHP з використанням MySQL бази даних. Платформа орієнтована на організацію взаємодії між викладачем та учнями та підходить як для організації дистанційних курсів так і підтримки очного навчання.

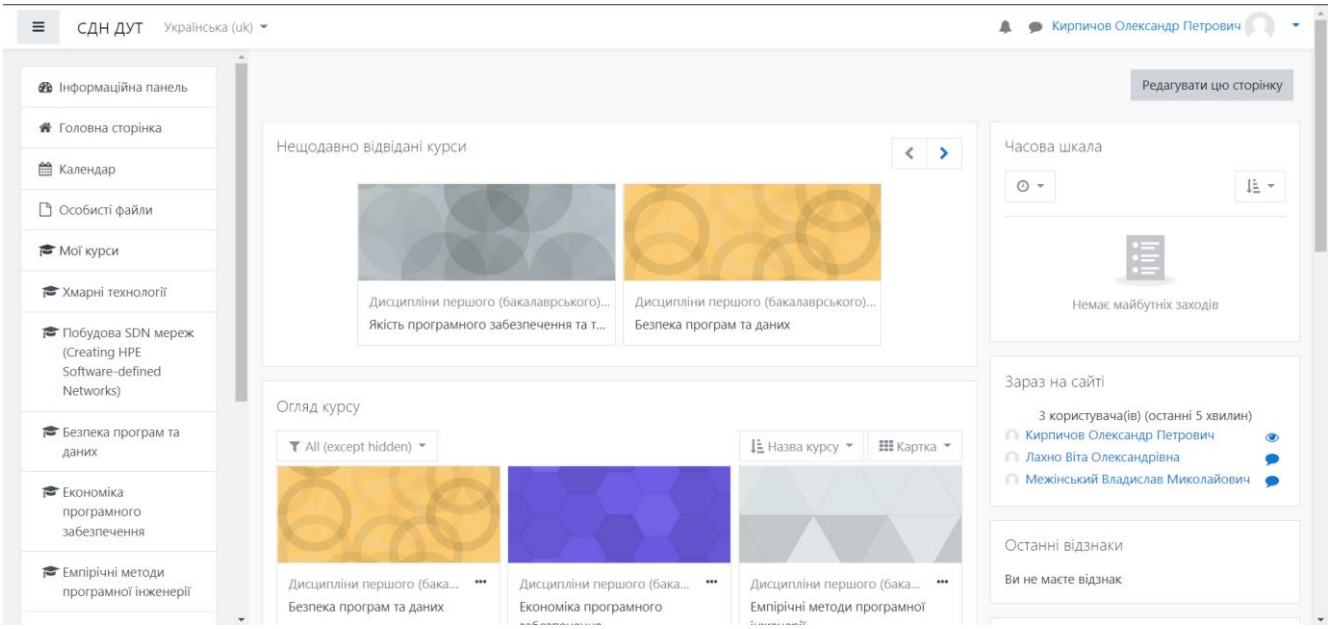


Рисунок 4. - Головна сторінка Moodle (учень)

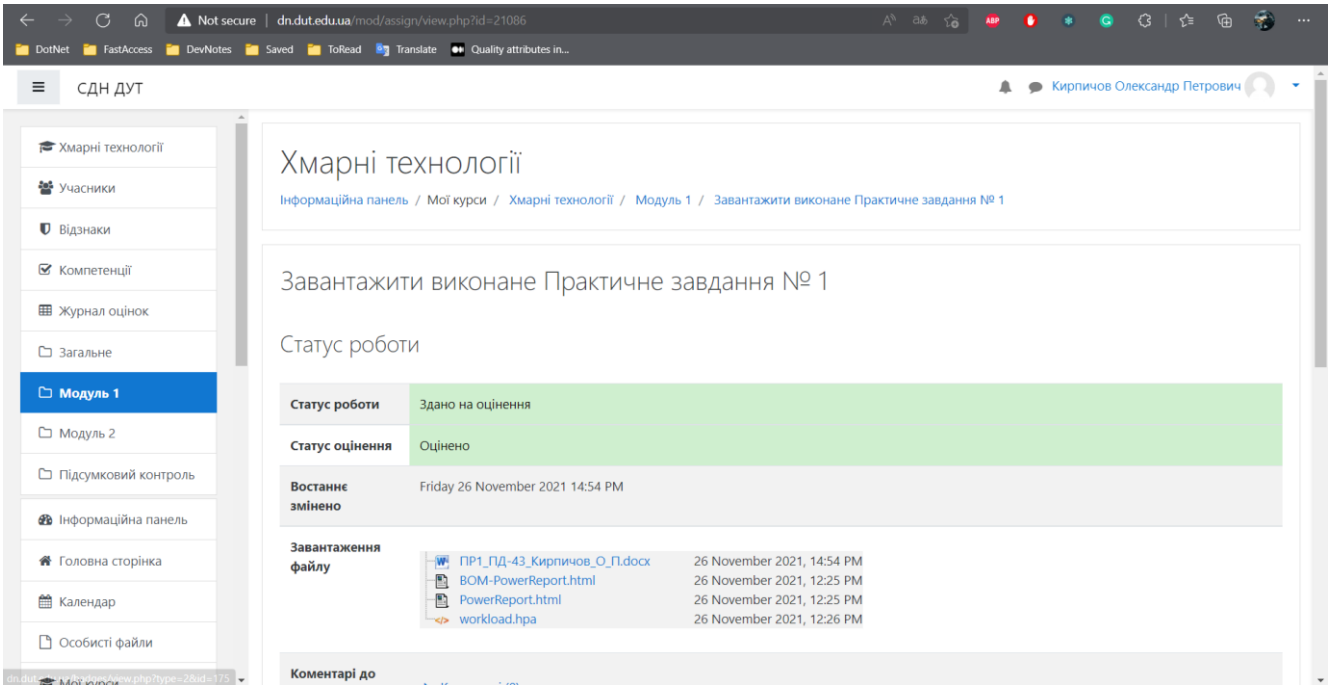


Рисунок 5. - Виконання завдання на Moodle (учень)

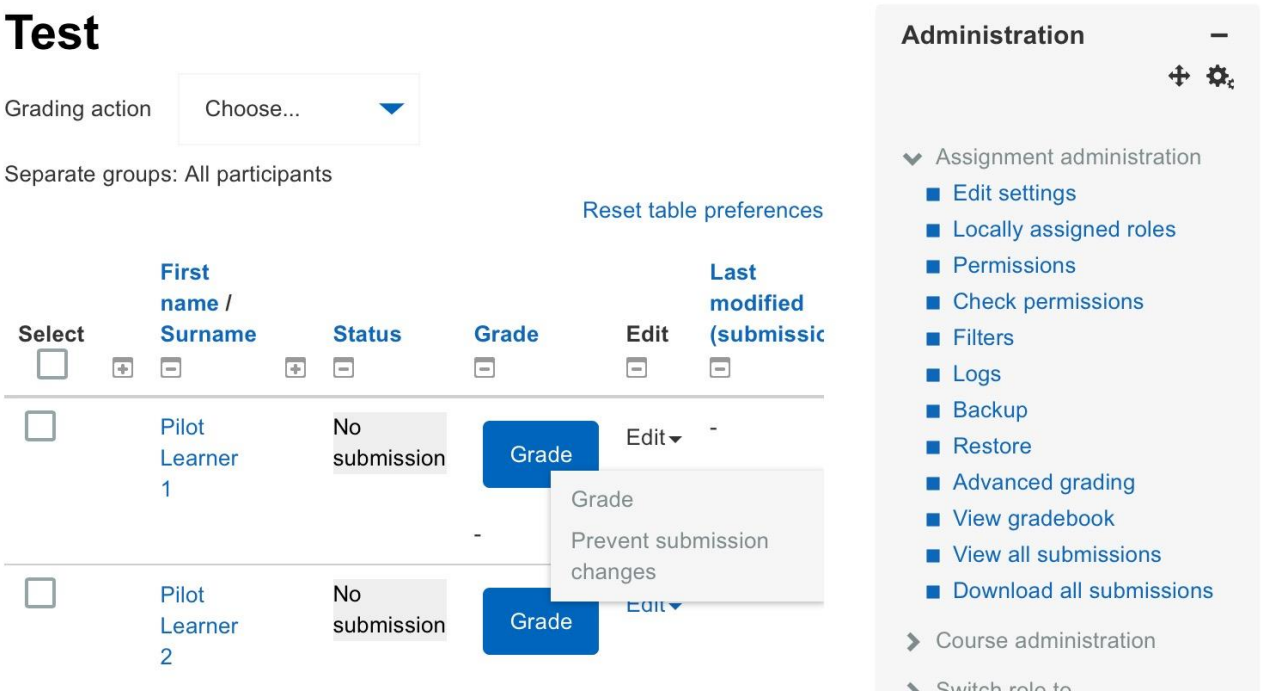


Рисунок 6. - Оцінка робіт в Moodle (викладач)

Переваги Moodle:

- Відкритий початковий код
- Платформа є екосистемою для учбового закладу
- Внутрішня система новин та сповіщень в межах платформи
- Наявність панелі керування для викладача
- Наявність панелі керування для адміністратору
- Можливість створення завдань та прикріплення файлів вкладень
- Можливість залишати персональні коментарі до завдань
- Можливість оцінки завдань та перегляд оцінок учнем
- Система керування ролями користувачів
- Підтримка плагінів (можливість кастомізації)

Недоліки Moodle:

- PHP є відносно старою технологією
- Відсутність мобільного додатку
- Орієнтованість на викладача та освітній заклад

- Відсутність статистики складності завдань та загального прогресу групи
- Відсутність можливості отримання сповіщень через месенджер
- Відсутність можливості отримання сповіщень через електронну пошту, яка не належить домену Moodle
- Відсутність підтримки користувацьких вебхуків
- Відсутність можливості створення динамічної головної сторінки класу
- Відсутність підтримки розкладу онлайн занять
- Відсутність користувацького TODO листу

1.3. Trello

Trello – це хмарне рішення створене Fog Creek Software, яке потім було куплено компанією Atlassian, яка є розробником Jira. Сервіс представляє з себе інтерактивну дошку для керування проектами для невеличких груп користувачів.

Безкоштовна версія додатку дозволяє створювати до 10 дошок для команди, необмежену кількість карток, необмежену кількість файлів об’ємом до 10МБ, кастомізувати дошку користувацькими фонами та наліпками, створювати дедлайни, нагадування та встановлювати виконавців для карток. Усі користувачі в такій версії мають рівні права, що може ускладнювати керування дошкою. Користувач може створювати необхідні йому колонки та додавати туди картки з необхідною інформацією, вкладенням та мітками (до 10 в безкоштовній версії) з можливістю їх переміщення, редагування та видалення. Також користувачі мають можливість залишати коментарі під картками.

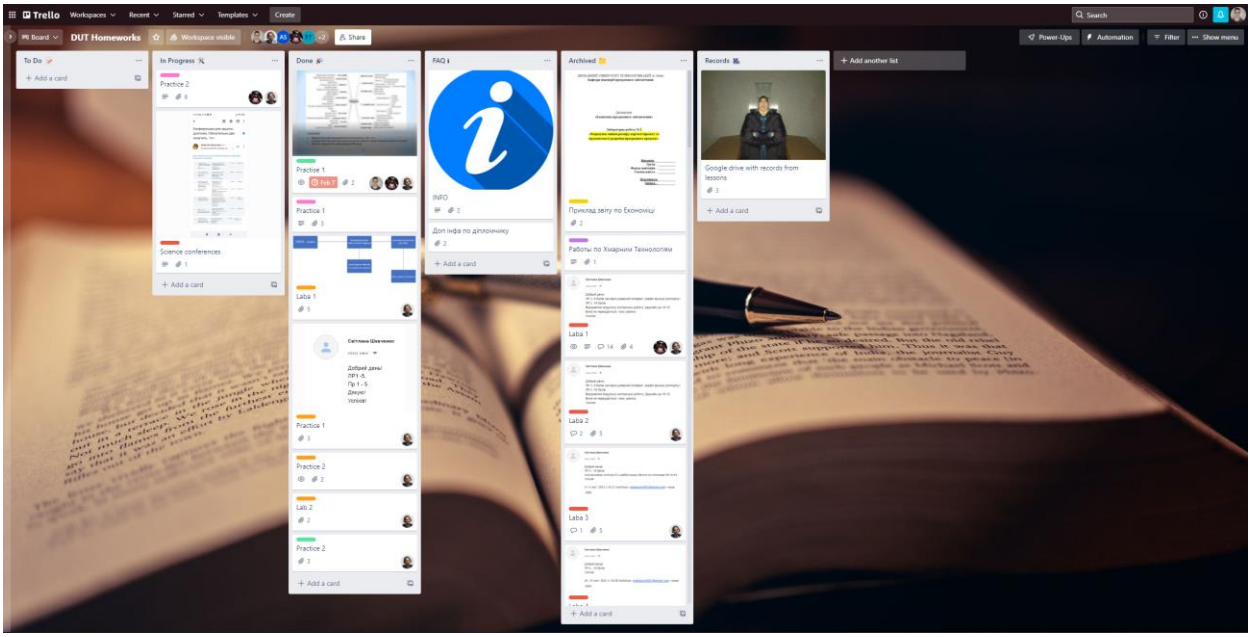


Рисунок 7.- Дошка в Trello

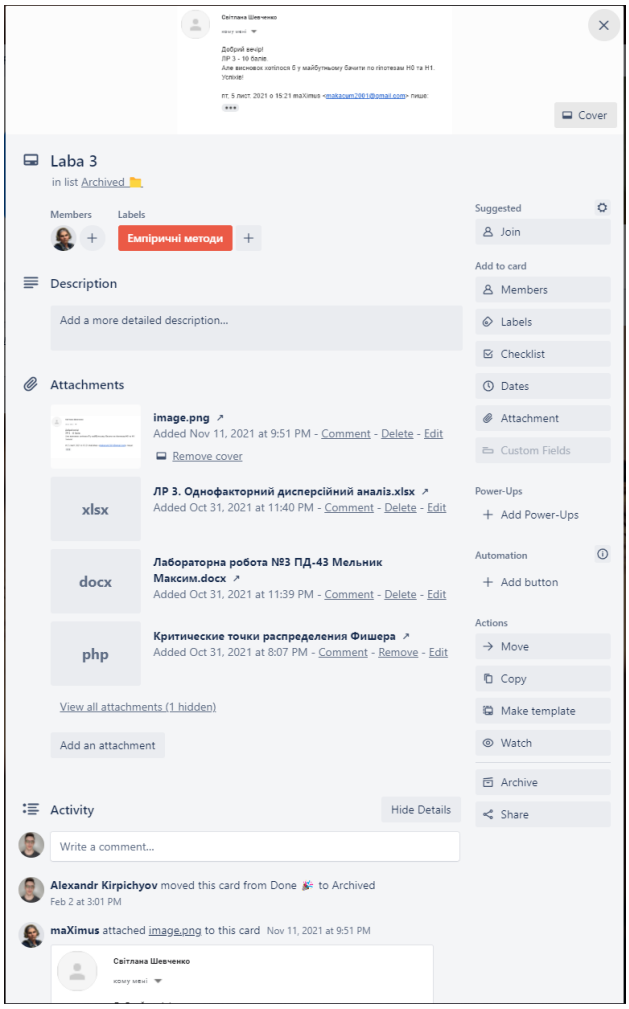


Рисунок 8. - Картка в Trello

Переваги Trello:

- Гнучка система дошок
- Внутрішня система сповіщень в межах платформи та засобами електронної пошти
- Наявність панелі керування доступами
- Підтримка мобільних пристроїв
- Можливість створення карток та прикріплення файлів вкладень
- Можливість залишати коментарі до карток
- Система керування ролями користувачів (платна версія)
- Підтримка плагінів (можливість кастомізації)
- Відкрите API
- Підтримка користувацьких вебхуків

Недоліки Trello:

- Лімітованість безкоштовної версії
- Закритий початковий код
- Загальна орієнтованість (не є інструментом розробленим саме для освітнього процесу)
- Відсутність статистики складності завдань та загального прогресу групи
- Відсутність можливості отримання сповіщень через месенджер
- Відсутність можливості створення динамічної головної сторінки класу
- Відсутність підтримки розкладу онлайн занять

1.4. Висновки

За результатами аналізу аналогів можна зробити висновок, що вони мають значну кількість переваг, але при цьому є певні спільні недоліки, усунувши які можна отримати дійсно зручне рішення для студентів. Складемо таблицю (див.

табл. 1) для порівняння аналогів з Assist Easy, де “+” – це наявність функціоналу, “-” – відсутність, а “*” часткова наявність.

Таблиця 1. – Порівняння аналогів з Assist Easy

	Google Classroom	Moodle	Trello	Assist Easy
Відкритий початковий код	-	+	-	+
Відкритий API	+	*	+	+
Панель керування викладача	+	+	-	-
Панель керування учня	+	+	+	+
Панель керування адміністратора	-	+	*	+
Можливість створення та редагування завдань	+	+	+	+
Можливість прикріплення файлових вкладень	+	+	+	+
Можливість коментування завдань	+	+	+	+
Персональна статистика	*	*	-	+
Групова статистика	-	-	-	+
Можливість перегляду оцінок учнем	+	+	-	-
Можливість отримання сповіщень через Email	+	*	+	+
Можливість отримання сповіщень через Telegram	-	-	-	+
Підтримка користувацьких вебхуків	-	-	+	+
Динамічна головна сторінка групи	-	-	-	+
Підтримка розкладу онлайн занять	-	-	-	+
Користувацький TODO лист	*	*	+	+
Можливість самостійного створення облікового запису	+	-	+	+

	Google Classroom	Moodle	Trello	Assist Easy
Керування власним акаунтом	+	-	+	+

Під час аналізу предметної галузі та аналогів, виявлено, що розроблюваний продукт немає абсолютних аналогів, тому в даному розділі було розглянуто продукти, які мають схожий функціонал та галузь використання. За результатами порівняння було складено таблицю, яка описує ключовий функціонал, що повинен бути реалізований.

2. ВИМОГИ ТА ОЦІНКА ЯКОСТІ СИСТЕМИ

2.1. Функціональні вимоги

2.1.1. Аутентифікація

Користувач повинен мати можливість зареєструватися у веб-сервісі та війти в свій обліковий запис. Для створення запису повинні використовуватися ім’я та прізвище, електрона пошта і пароль. Вхід в систему повинен здійснюватися за допомогою email та паролю. У випадку, якщо користувач забув свій пароль – повинна бути можливість встановити новий за допомогою спеціального посилання, надісланого через електрону пошту.

2.1.2. Обліковий запис користувача

Зареєстрований користувач повинен мати можливість підтвердити свою електрону поштову адресу, а також бачити та змінювати наступні дані свого облікового запису:

- Ім’я та прізвище
- Аватар
- Пароль (використовуючи свій поточний пароль)
- Налаштування повідомлень
    - Отримання повідомлень через email
    - Отримання повідомлень через telegram
    - Значення, за скільки годин треба повідомити про дедлайн завдання
    - Значення, за скільки годин треба повідомити про онлайн заняття

2.1.3. Групи

Користувач повинен мати можливість створити групу, використовуючи назву та зображення, та керувати нею, або приєднатися до існуючої групи за допомогою унікального коду, надісланого через email. Учасники групи повинні мати можливість переглядати весь контент який там знаходиться. Кожна група

повинна мати можливість переглядати та редагувати динамічну сторінку за допомогою розмітки markdown. Студент може одночасно перебувати лише в одній групі, а для створення чи приєднання до групи потрібно мати підтверджену email адресу. Учасник групи може в будь який момент покинути групу, що повинно призвести до видалення його завдань та сповіщень.

2.1.4. Ролі в групі

Учасники групи повинні мати одну з трьох ролей, що наділяють учнів певним рівнем доступу. Ролі та відповідні можливості представлені матрицею (див таблицю 2).

Таблиця 2. – Матриця ролей та можливостей в системі

	Власник	Адміністратор	Член групи
Перегляд контенту групи	+	+	+
Створення оголошень	+	+	-
Керування ролями студентів групи	+	-	-
Редагування динамічної сторінки групи	+	-	-
Редагування списку предметів	+	+	-
Редагування списку завдань з предметів	+	+	-
Редагування розкладу онлайн занять	+	+	-
Запрошення учасників в групу	+	-	-
Видалення учасників з групи	+	-	-
Видалення групи	+	-	-
Керування власним TODO листом	+	+	+
Керування власними веб хуками	+	+	+

2.1.5. Предмети

Система повинна забезпечувати можливість перегляду, створення, редагування та видалення предметів групи. Предмет групи повинен включати в себе наступну інформацію: назва, опис, ім’я та прізвище викладача, колір та іконка.

2.1.6. Завдання

Система повинна забезпечувати можливість перегляду, створення, редагування та видалення завдань з предмету групи. Завдання групи повинно включати в себе наступну інформацію: назва предмету, назва завдання, опис, тип завдання (домашнє завдання, лабораторна, тестування, лекція, практична, інше), дата здачі завдання (дедлайн) та файлові вкладення. Повинна бути забезпечена підтримка пагінації та сортування за датою здачі.

Користувачі повинні мати можливість залишити, редагувати та видалити свій коментар до завдання. Інші користувачі повинні бачити коментарі один одного, що включають в себе: текст, аватар, дату написання, а також ім’я та прізвище автора.

2.1.7. Розклад онлайн занять

Система повинна забезпечувати можливість перегляду, створення та видалення онлайн занять. Онлайн заняття повинно включати в себе наступну інформацію: посилання на заняття, назва, опис, назва предмету, дата та час наступної пари, а також періодичність в днях. Після того, як заняття відбулося – дату наступного заняття повинно бути оновлено автоматично, враховуючи налаштовану для заняття періодичність.

2.1.8. TODO лист

Система повинна забезпечувати можливість перегляду, створення та оновлення статусу виконання елементів TODO листа. Елемент листа повинен включати в себе наступну інформацію: статус (завершено, не завершено), назва завдання, назва предмета, дата виконання та дата наступного сповіщення.

Користувач повинен мати можливість створювати, редагувати час та видаляти сповіщення для елементу листу.

2.1.9. Користувацькі вебхуки

Система повинна забезпечувати можливість перегляду, реєстрації та видалення користувацьких вебхуків. Вебхук включає в себе секретний код, URI та тип події (завдання було додано або завдання було оновлено).

Після реєстрації вебхуку та в разі виникненні відповідної до типу хуку події, запит з інформацією про подію та кодом хуку повинно бути надіслано за посиланням на вказаний URI ресурс.

2.1.10. Рейтинг складності завдань

Система повинна забезпечувати можливість створення та редагування відгуку про складність завдання користувачем. Відгук складається з оцінки складності від 1-5 включно та часу витраченого на виконання в хвилинах. На основні відгуків користувачів повинна формуватися інформацію про середню складність завдання, яку можуть бачити усі користувачі групи.

2.1.11. Статистика користувача

Система повинна автоматично формувати корисну статистику для користувача, що включає в себе наступну інформацію:

- Загальна кількість завдань для виконання
- Кількість завдань в TODO листі
- Кількість виконаних завдань
- Середня оцінка складності завдань
- Середній час виконання завдань

2.1.12. Система сповіщень

Система повинна забезпечувати роботу системи сповіщень для користувачів групи, яка доставляє повідомлення до студентів згідно налаштувань повідомлень студента та типу сповіщення. Повинні підтримуватися такі засоби доставлення сповіщень, як електронна пошта та Telegram. Можливість отримання сповіщень повинні мати лише учні, що мають підтверджені акаунти на відповідних ресурсах чи платформах, що використовуються для доставки повідомлень.

Користувач отримує повідомлення з відповідним вмістом при наступних подіях:

- Було додано нове завдання
- Існуюче завдання було оновлено
- Згідно до налаштувань користувача, дедлайн виконання завдання наближається
- Згідно до налаштувань користувача, дата проведення онлайн заняття наближається
- Настав час, налаштований користувачем, для нагадування про завдання в TODO листі
- Було опубліковано оголошення
- Роль користувача в групі було змінено
- Користувача було виключено з групи
- Користувача було запрошено до групи

2.1.13. Діаграми використання

Також вимоги до функціоналу додатково представлені у виді діаграми використання (див. рис. 9-13).

Рисунок 9. – Діаграма використання - адміністратор (керування групою)

Рисунок 10. – Діаграма використання - адміністратор (керування предметами)

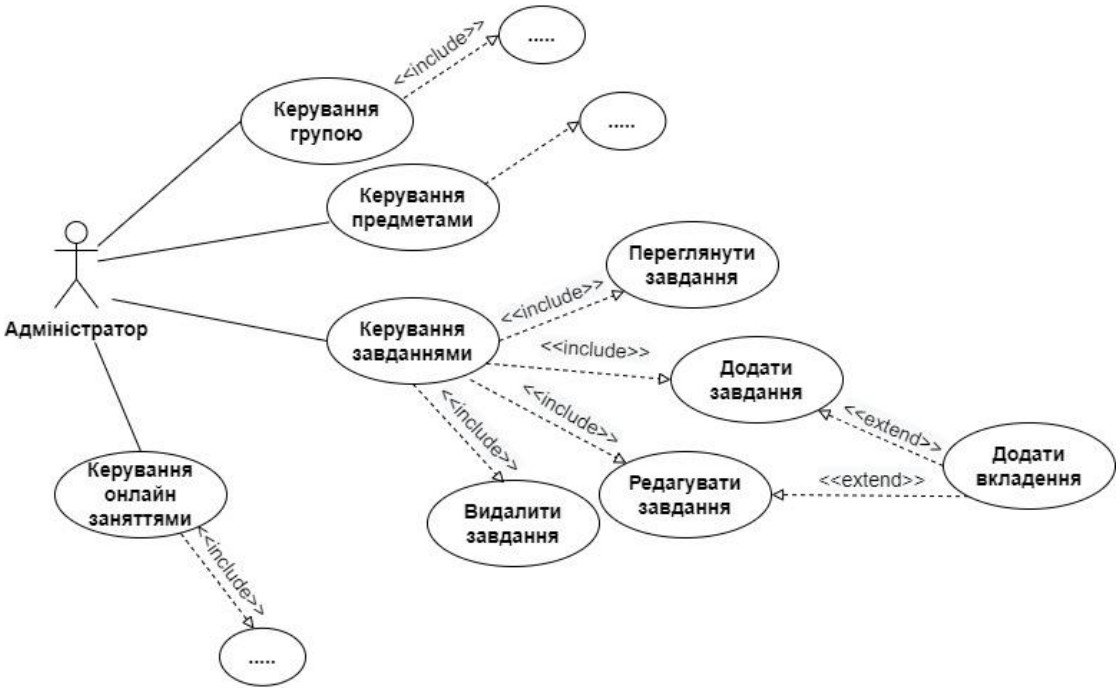


Рисунок 11. – Діаграма використання - адміністратор (керування завданнями)

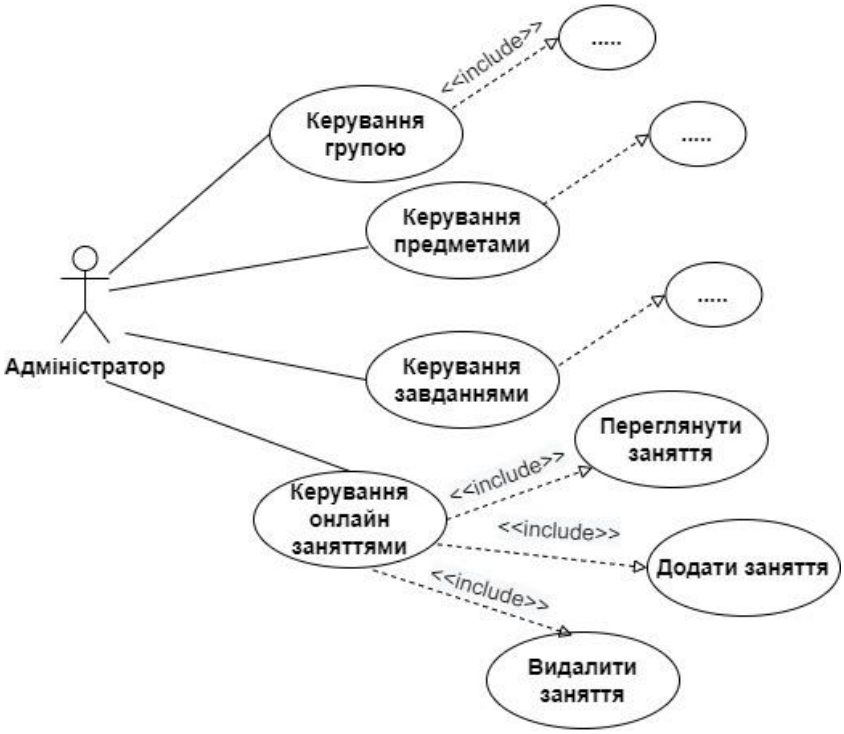


Рисунок 12. – Діаграма використання - адміністратор (керування онлайн заняттями)

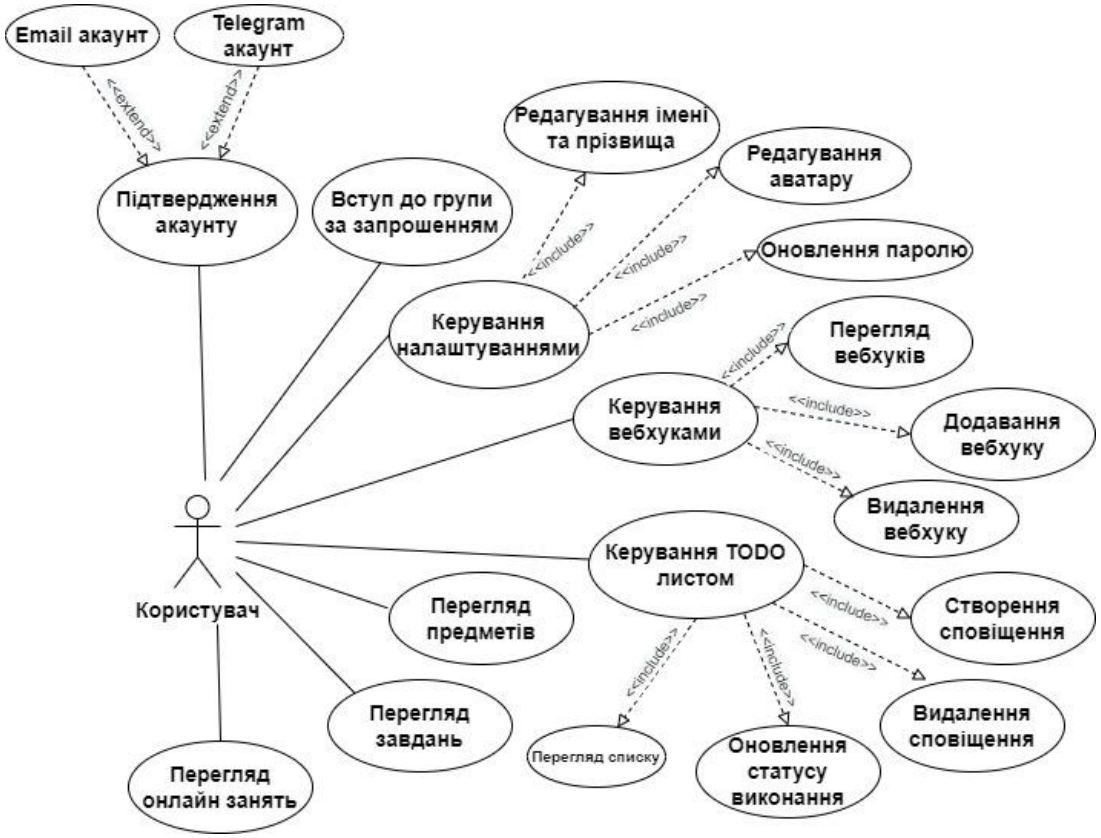


Рисунок 13. – Діаграма використання - користувач

2.2. Нефункціональні вимоги

2.2.1. Продуктивність

Продуктивність програмного продукту AssistEasy визначається згідно до встановлених в цьому розділі метрик щодо швидкості виконання окремих запитів та загальної пропускної здатності сервісу. Система повинна витримувати навантаження, описані в таблиці нижче (див. табл. 3)

Таблиця 3. – Вимоги до витримування навантажень

Категорія	Кількість запитів / одиниця часу
Запити від будь якого користувача (операція читання)	10,000 / хвилину
Запити від одного користувача (операція читання)	10 / секунду

Категорія	Кількість запитів / одиниця часу
Запити від одного користувача (операція створення)	5 / секунду
Запити від одного користувача (операція редагування)	3 / секунду
Запити від одного користувача (операція видалення)	2 / секунду
Відправлення повідомлень	120 / секунду

2.2.2. Сумісність

Програмний продукт повинен задовольняти наступним вимогам сумісності:

- API веб сервісу повинно мати підтримку версіонування
- API контракт повинен підтримувати HTTP/S запити в форматі JSON
- Система повинна бути сумісна з бібліотеками TelegramBot API та MailKit
- Програмне забезпечення повинно бути сумісним з ОС Linux (CentOS, Debian, Fedora, RHEL, SLES, Ubuntu) та Windows (Server 2016, Server 2019)

2.2.3. Зручність використання

Сервіс повинен задовольняти наступним вимогам зручності використання:

- API повинен включати в себе документацію із списком ендпоінтів та їх моделей
- API повинно підтримувати пагінацію та сортування
- API повинно слідувати принципам RESTful
- API повинно повертати наступні HTTP коди:
    - 200 в разі успішного запиту та наявності контенту в відповіді
    - 204 в разі успішного запиту та відсутності контенту в відповіді
    - 400 в разі наявності помилок у моделі запиту
    - 401 в разі отримання запиту від неавтентифікованого запиту
    - 403 в разі отримання запиту на доступ до ресурсу, що не підпадає під його рівень доступу

    - 404 в разі відсутності ресурсу для користувача
- Інтерфейс веб клієнта повинен мати діалогові або popup повідомлення або відображати loader при виконанні запиту до API
- Інтерфейс веб клієнта повинен використовувати червоний колір та вікно підтвердження для дій пов’язаних з видаленням ресурсу
- Інтерфейс веб клієнта повинен мати меню з швидким доступом до сторінок групи та налаштувань
- Веб клієнт повинен підтримувати наступні розширення екрану:
    - 1920x1080
    - 1366×768
    - 1536×864

2.2.4. Надійність

Система повинна задовольняти наступним вимогам надійності:

- Система повинна оброблювати можливі рантайм помилки
- Система повинна продовжувати приймати запити та відповідати на них при виникненні неопрацьованих помилок
- Система повинна продовжувати приймати запити та відповідати на них при виникненні проблем із з’єднанням з джерелом даних
- Система повинна оброблювати та валідувати моделі запитів від користувачів
- Програмні модулі, які мають залежність від мережевих чи файлових ресурсів, повинні вивільняти пам’ять та закривати усі з’єднання після завершення роботи модулю

2.2.5. Доступність

Система повинна задовольняти наступним вимогам доступності, що описані в таблиці нижче (див. табл. 4).

Таблиця 4. – Значення вимог доступності

Категорія	Значення
Максимальний downtime	(0.5%) = 3.36 год. / місяць
Мінімальний uptime	(95.5%) = 641.76 год. / місяць
Максимальний downtime під час розгортання оновлення	1 год.
Максимальний downtime під час відкату оновлення	1 год.

2.2.6. Безпека

Система повинна задовольняти наступним вимогам безпеки:

- Паролі користувачів повинні зберігатися у вигляді хешу
- Веб сервіс повинен використовувати механізм автентифікації та авторизації, що дозволяє миттєво застосовувати оновлення інформації щодо доступу до ресурсів системи
- Програмні модулі, що взаємодіють з джерелом даних, повинні валідувати запити на SQL-ін’єкції
- Програмні модулі, що приймають HTML або Markdown розмітку та оброблюють її, повинні проводити санітайзинг перед збереженням в джерело даних
- Користувачі не повинні мати доступ до персональних ресурсів один одного
- Усі ендпоінти повинні мати механізми для перевірки рівню доступу користувача в системі
- Frontend клієнти API повинні використовувати HTTPS протокол для здійснення запитів до сервісу

2.2.7. Ремонтопридатність та підтримка

Система повинна задовольняти наступним вимогам ремонтопридатності та підтримки:

- Сервіс повинен використовувати логування з подальшою можливістю перегляду логів, які повинні включати в себе дату й час події у форматі UTC, тип події (дебаг, інформація, попередження, помилка, критична помилка), опис події та інформацію про виключення, включаючи стектрейс (якщо виключення присутнє)
- Система повинна мати механізм для перевірки стану роботи мікросервісів (ступінь навантаження та індикатор доступності мікросервісу)
- Модулі програми повинні мати високий показник щільності та низький показник залежності
- Система повинна мати механізм для швидкого розгортання патчів та оновлень на продакшн сервер

2.2.8. Можливість модифікації

Програмні модулі системи повинні використовувати dependency injection, шаблони проектування та слідувати принципам SOLID, KISS, DRY та YAGNI. Повторюванні елементи системі повинні бути винесені в окремі методи, класи або бібліотеки.

2.2.9. Тестування

Мінімальний показник покриття для програмних модулів системи складає 5%. Покриття повинно бути забезпечене unit або integration тестами. Окрім використання тестів для програмних модулів, повинно бути забезпечене 100% покриття системи ручним або автоматизованим тестуванням.

2.2.10. Масштабованість

Усі мікросервіси системи повинні підтримувати як горизонтальне так і вертикальне масштабування.

2.2.11. Вимоги у вигляді асоціативної мапи

Оскільки використання асоціативних мап дозволяє легко й наочно представити необхідно інформацію та дозволяє легко проаналізувати основні зв’язки між компонентами мапи, вимоги додатково були представлені у вигляді Mind map (див. рис. 14-15).

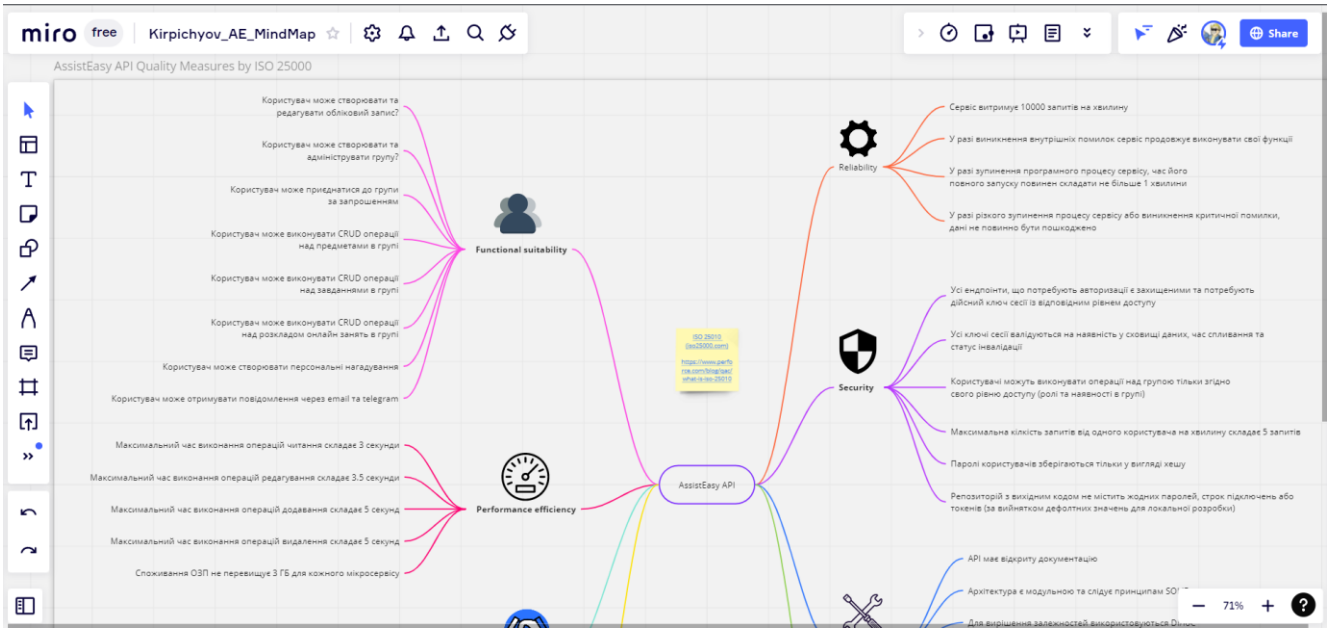


Рисунок 14. – Асоціативна мапа вимог до AssistEasy

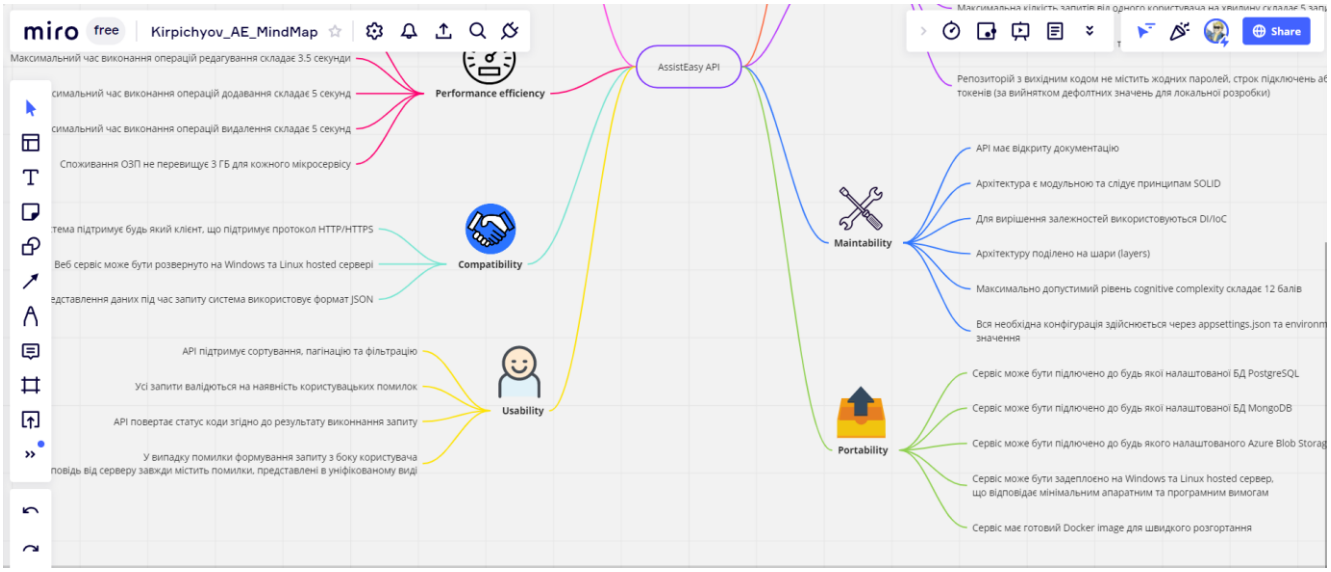


Рисунок 15. – Асоціативна мапа вимог до AssistEasy

2.3 Системні вимоги

Для забезпечення коректної роботи системи, серверна частина повинна мати встановлений Docker та мати виділені фізичні або віртуальні апаратні ресурси згідно до рекомендованих системних вимог наведених нижче (див. табл. 5).

Таблиця 5. – Системні вимоги до сервера

Мікросервіс / системний компонент	Процесор	Оперативна пам'ять	Вільне місце на диску	ОС
Core	2.2 GHz 2 ядра 2 потоки	1024 МБ DDR3	256 МБ HDD	Windows 10+ / Windows Server 2016+ / Linux (Ubuntu 18.04+, Ubuntu Server, Debian, OpenSUSE,
MailSender	1.4 GHz 1 ядро 1 потік	256 МБ DDR3	256 МБ HDD	
Scheduler	2.2. GHz 2 ядра 2 потоки	512 МБ DDR3	256 МБ HDD	
TelegramBot	1.4 GHz 2 ядра 2 потоки	256 МБ DDR3	256 МБ HDD	
WebhookSender	1.4 GHz 1 ядро 1 потік	256 МБ DDR4	256 МБ HDD	
PostgreSQL	2 GHz 2 ядра 2 потоки	2560 МБ DDR4	3072 МБ SSD	

Мікросервіс / системний компонент	Процесор	Оперативна пам’ять	Вільне місце на диску	ОС
MongoDB	2 GHz 2 ядра 2 потоки	2560 МБ DDR4	3072 МБ SSD	Fedora Server)
RabbitMQ	1.4 GHz 1 ядро 1 потік	4096 МБ DDR4	4096 МБ HDD	

Системні вимоги до клієнту залежать від реалізації front-end частини програмного забезпечення, а оскільки основною метою є розробка серверної частини – відповідні вимоги до клієнту опущено.

3. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ВЕБ-СЕРВІСУ

3.1. Інструменти для розробки

3.1.1. Docker

Docker – це програмна платформа для швидкої розробки, тестування та розгортання додатків. Він складає програмне забезпечення в контейнери, які містять у собі усі необхідні компоненти для його роботи (бібліотеки, SDK, код, файли та середу виконання). AssistEasy використовує докер для налаштування локальної середи розробки за допомогою docker compose файлу, який містить у собі конфігурацію для розгортання середи. Після виконання відповідної команди файл зчитується, та докер розгортає середу, що містить у собі БД PostgreSQL та MongoDB, а також брокер повідомлень RabbitMQ (див. рис. 16-17).

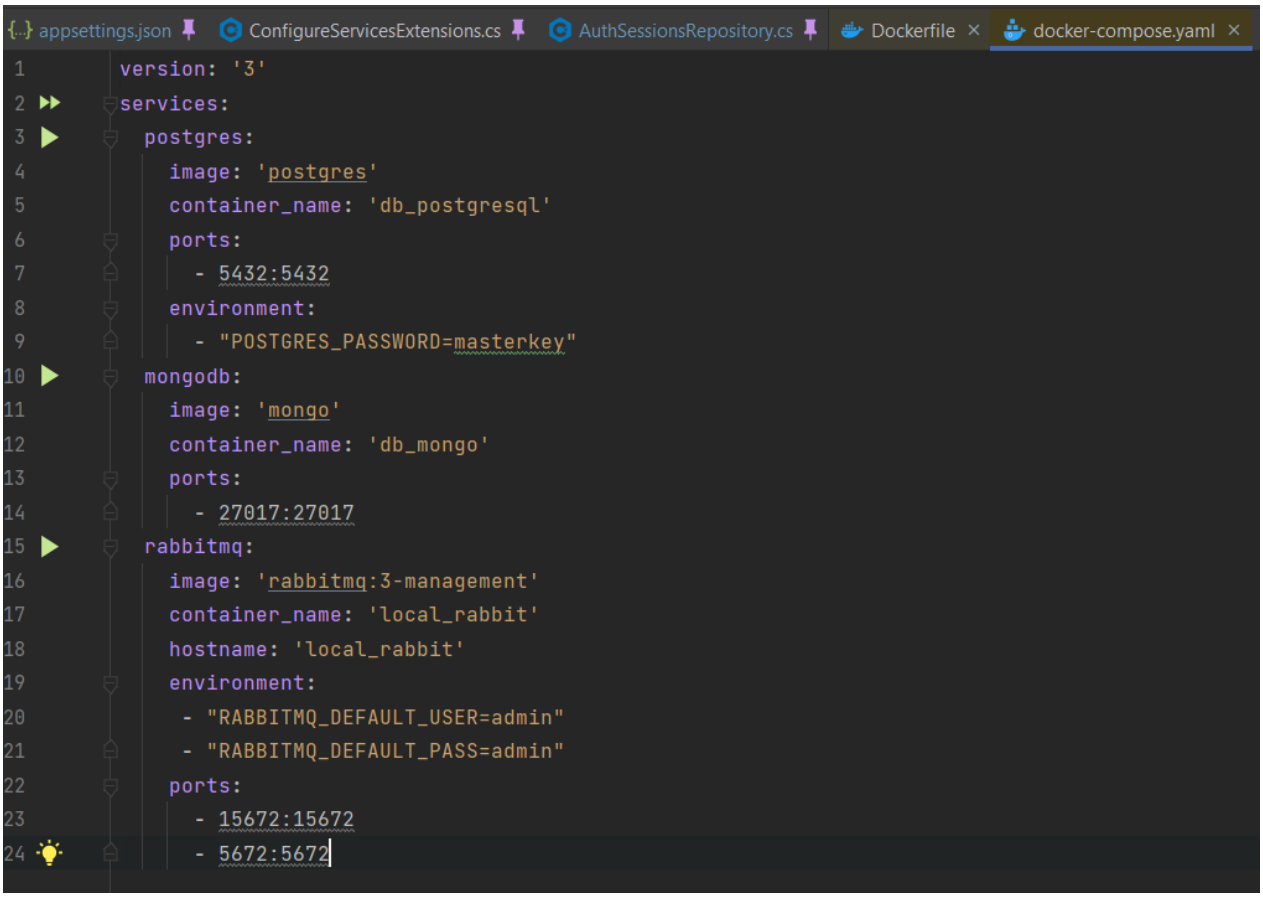


Рисунок 16. – Файл для розгортання середи

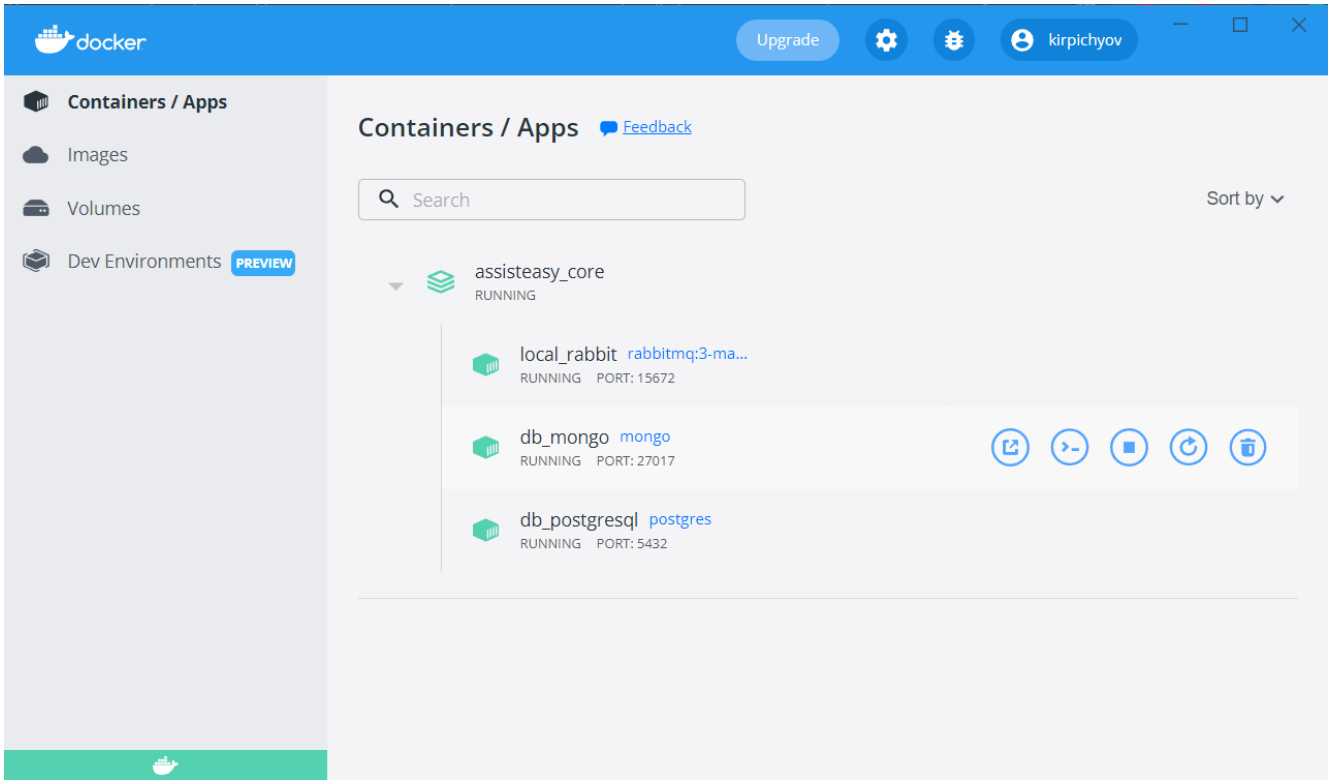


Рисунок 17. – Розгорнута середа в Docker

3.1.2. Git

Git – це дуже потужна розподілена система керування версіями файлів. Вона дозволяє легко слідкувати за змінами в програмному продукті, вносити нові, створювати гілки, повертатися до старих версій та розроблювати продукт в команді. Більшість IDE має зручний візуальний інтерфейс для роботи з системою контролю версій, але для розробки веб сервісу використовувався Git Bash та консоль Windows Terminal (див. рис. 18).

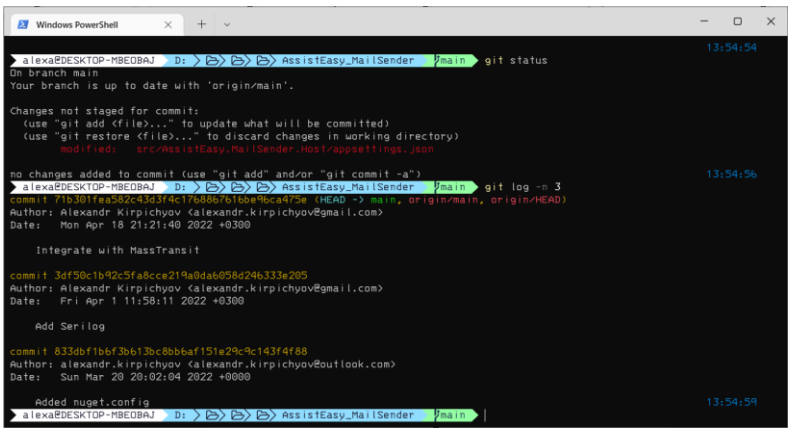


Рисунок 18. – Використання Git

3.1.3. Azure DevOps

Azure DevOps – це комплексне рішення корпорації Microsoft, що містить в собі систему керування версіями, інструменти розгортання додатку, тестування, аналізу коду та є потужною платформою для спільної роботи над розробкою програмного забезпечення. Усі репозиторії AsistEasy знаходяться в Azure DevOps (див. рис. 19) та використовують спеціально розроблений сценарій для перевірки коду при відкритті пул реквесту чи доданні змін у головну гілку. Цей сценарій має декілька кроків (див. рис. 20-21):

1. Налаштування середи, де будуть виконуватися перевірки
2. Авторизація менеджеру пакетів
3. Завантаження пакетів проекту
4. Збірка проекту
5. Запуск unit тестів
6. Аналіз покриття та повідомлень при збірці

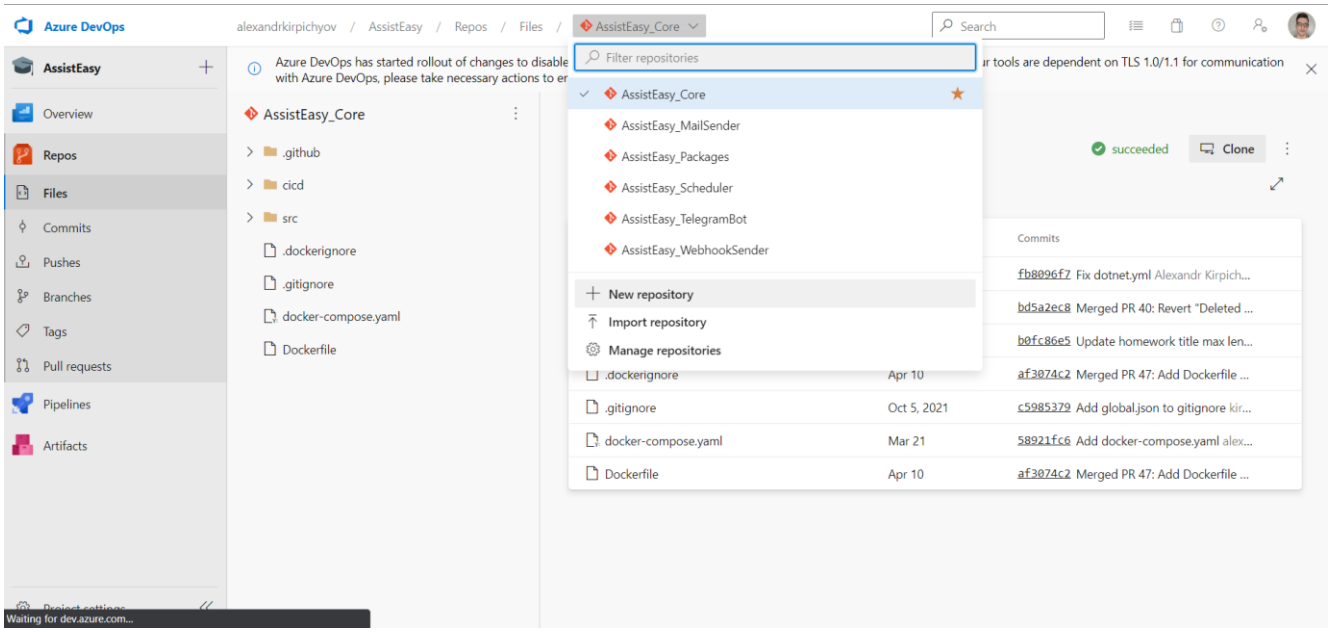


Рисунок 19. – Репозиторії AssistEasy в Azure DevOps

main / cicd / build_on_pull_request.yml

build_on_pull_request.yml

Edit

Contents History Compare Blame

```yaml
# ASP.NET Core (.NET Framework)
# Build and test ASP.NET Core projects targeting the full .NET Framework.
# Add steps that publish symbols, save build artifacts, and more:
# https://docs.microsoft.com/azure/devops/pipelines/languages/dotnet-core

name: 'Build on PR'

pool:
  vmImage: 'ubuntu-latest'

variables:
  solution: '**/*.sln'
  buildPlatform: 'Any CPU'
  buildConfiguration: 'Release'

steps:
- task: NuGetAuthenticate@0
  inputs:
    nuGetServiceConnections: 'AssistEasy_NuGet'

- task: NuGetToolInstaller@1

- task: NuGetCommand@2
  inputs:
    restoreSolution: '$(solution)'
    feedsToUse: 'config'
    nugetConfigPath: 'src/nuget.config'
    externalFeedCredentials: 'AssistEasy_NuGet'
  displayName: 'Restore Nuget Packages'
```

Рисунок 20. – Файл сценарію при доданні змін

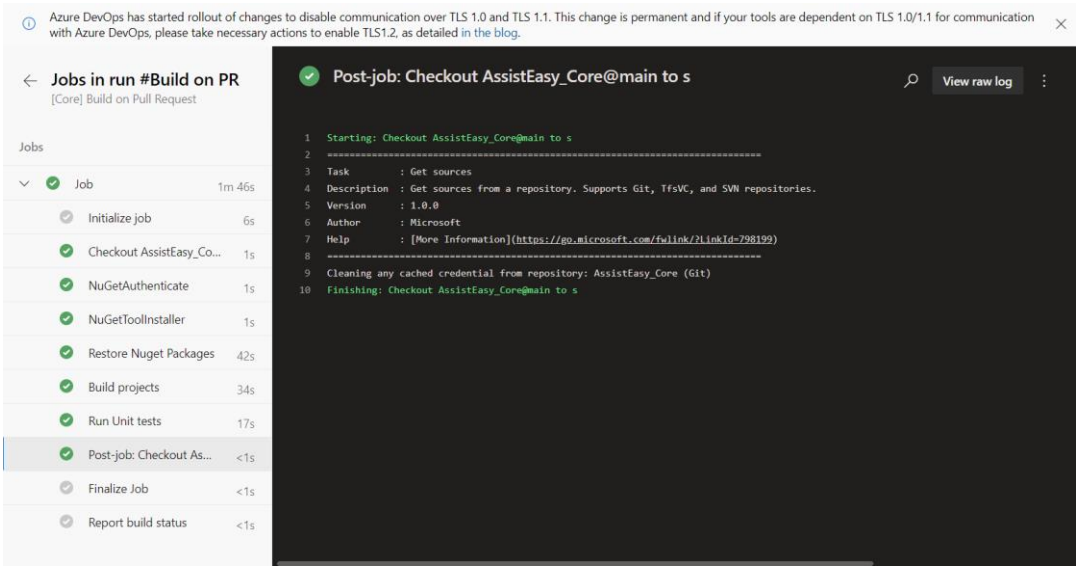


Рисунок 21. – Виконаний сценарій при доданні коміту

3.1.4. JetBrains Rider

Для розробки системи використовувалась провідна IDE Rider 2021.3 від компанії JetBrains. Вона має дуже потужний IntelliSense та засоби аналізу, рефакторингу та дебагу коду. Також в її складі є інструменти для перегляду IL коду та декомпіляції існуючих бібліотек. Є можливість складення TODO листів, закладок, менеджер брекпоінтів, переглядач баз даних та підтримка розширень (див. рис. 22).

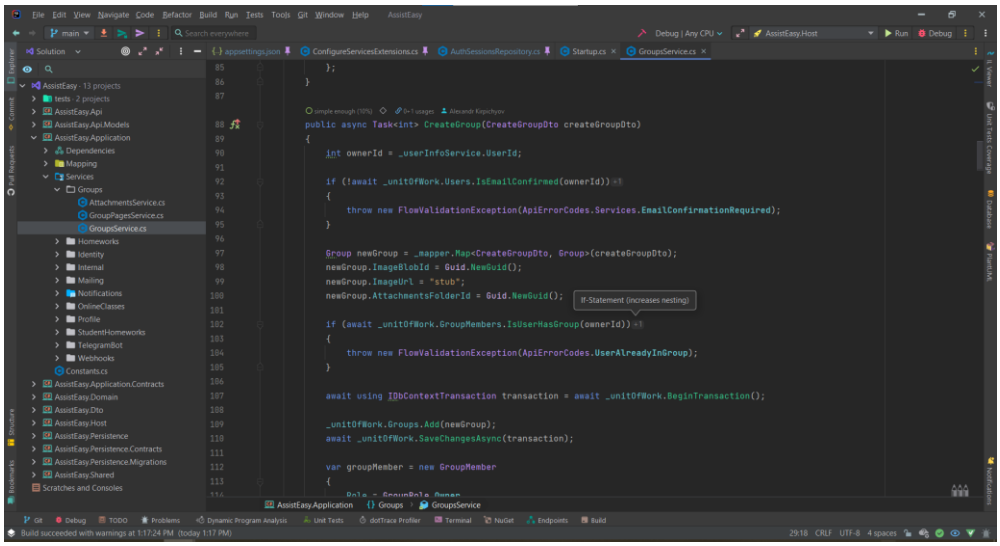


Рисунок 22. – Вікно IDE JetBrains Rider

3.1.5. DBeaver

Під час розробки програмного забезпечення, що використовує бази даних в якості сховища інформації, виникає потреба в перевірці наявності даних в таблицях, валідації схеми та виконання запитів. Зазвичай кожна СУБД має своєму складі програмне забезпечення для керування джерелом даних. Але набагато зручніше мати один інструмент, який дозволяє підключатися до багатьох різних БД одночасно та надає можливість керувати ними за допомогою єдиного інтерфейсу. Таким рішенням є додаток Dbeaver, який активно використовувався при розробці AssistEasy (див. рис. 23).

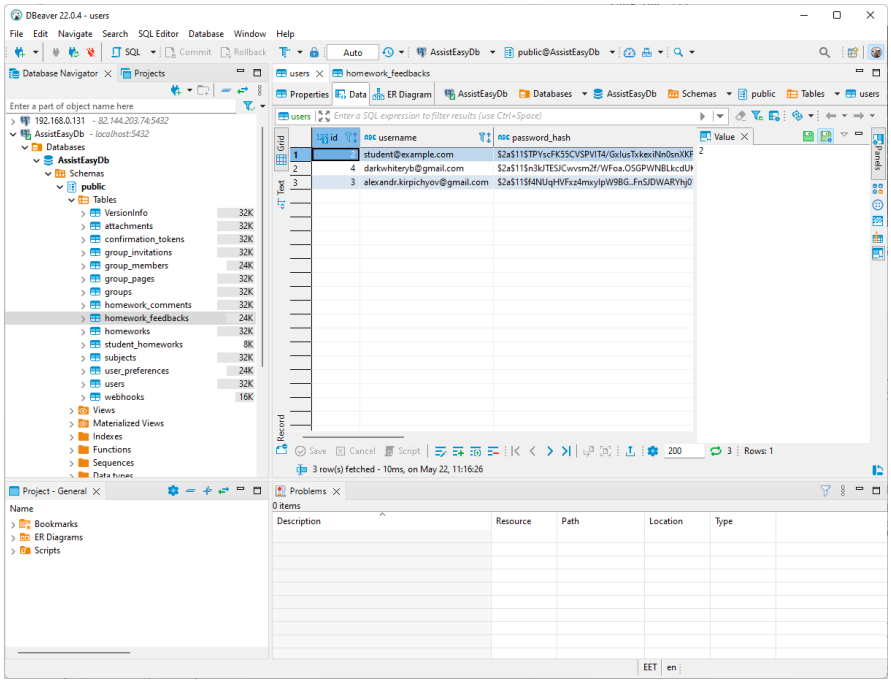


Рисунок 23. – БД AssistEasy в Dbeaver

3.2. Архітектура системи

Веб сервіс використовує мікросервісну архітектуру замість моноліту, оскільки цей підхід має кращі показники продуктивності, модифікованості та масштабованості. Архітектурний підхід на основі мікросервісів дозволяє:

- Краще розподіляти серверні ресурси
- Знизити рівень залежності програмних модулів один від одного
- Зробити систему більш гнучкою для змін
- Краще масштабувати сервіс
- Покращити показник когнітивної складності

Система складається із п’яти мікросервісів та використовує бази даних PostgreSQL та MongoDB в якості джерела даних. Для забезпечення швидкого обміну повідомленнями між мікросервісами використовується брокер повідомлень RabbitMQ. В якості сховища файлів вкладень та зображень використовується Azure BLOB storage. Загальну архітектуру зображено на схемі мікросервісів (див. рис. 24).

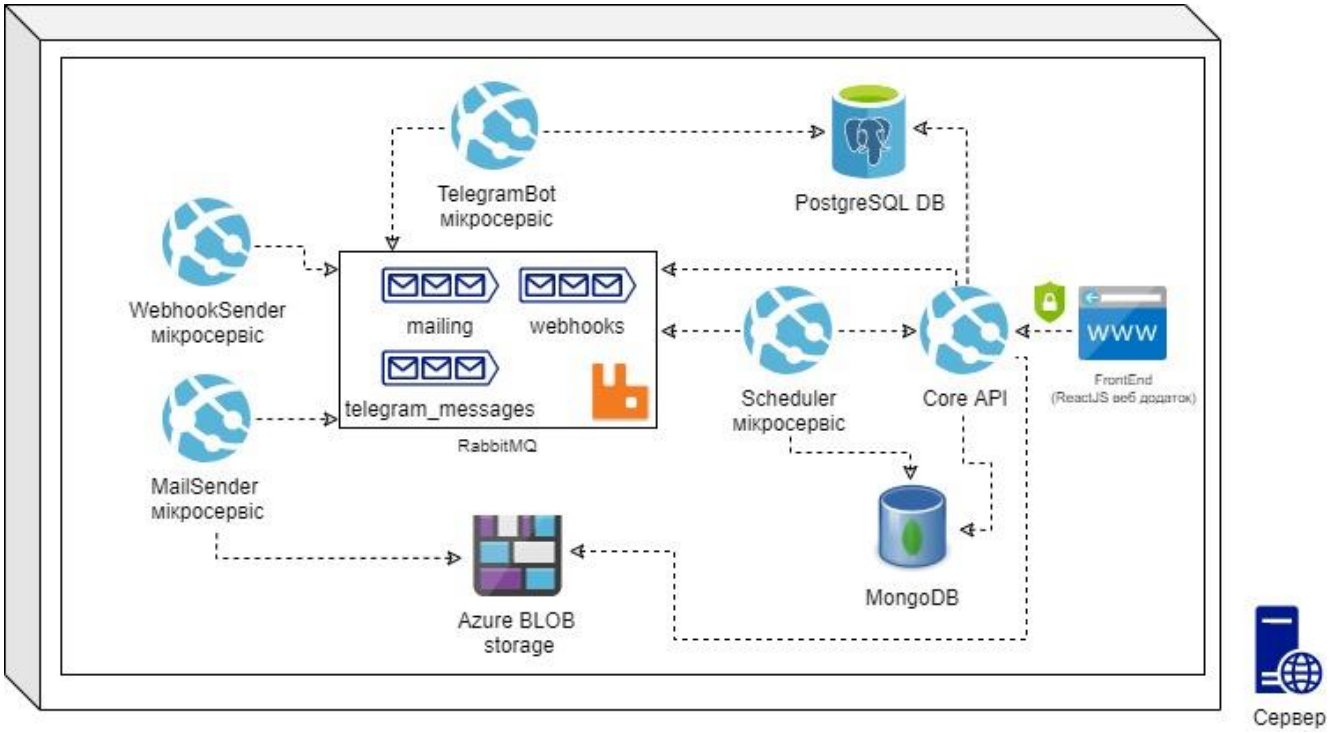


Рисунок 24. – Загальна архітектура мікросервісів

3.2.1. Реалізація мікросервісів на основі ASP.NET Core 5.0

Усі мікросервіси розроблено на базі фреймворку ASP.NET Core 5.0 мовою програмування C# версії 9.0 (.NET 5) у середовищі розробки JetBrains Rider.

ASP.NET Core – це крос-платформений та потужний фреймворк з відкритим кодом для створення сучасного хмарного веб додатку, який активно підтримується корпорацією Microsoft. Набір бібліотек ASP.NET дозволяє будувати прикладні програмні інтерфейси, сервіси, а також веб та IoT додатки. Також фреймворк має наступні переваги:

- Використовує потужну мову програмування C#
- Уніфікований підхід для створення веб-інтерфейсу та веб-API
- Архітектура дозволяє легко тестувати програмні компоненти
- Має у своєму складі технології Blazor та Razor Pages, які дозволяють легко розроблювати динамічні веб сторінки за допомогою мови програмування C#
- Розроблений додаток може працювати на Windows, macOS та Linux
- Відкритий код дозволяє швидко розробляти та покращувати фреймворк
- Потужна інтеграція з хмарними сервісами та екосистемою Microsoft

- Має вбудовану зручну систему конфігурування
- Має вбудовану потужну реалізацію Dependency injection
- Має вбудовану реалізацію шаблону MVC
- Підтримує RPC
- Є легким та швидким
- Може оброблювати до 7 млн. запитів в секунду
- Має модульну систему обробки HTTP запиту на основі фільтрів та middleware
- Підтримує хостинг за допомогою найпопулярніших технологій:
    - Kestrel
    - IIS
    - HTTP.sys
    - Nginx
    - Apache
    - Docker

3.2.2. PostgreSQL та MongoDB в якості джерел даних

Реляційні (SQL) та NoSQL рішення є різними типами баз даних, які мають принципово різну будову, зберігають дані по різному, а доступ до даних здійснюється за допомогою своїх мов запитів, які теж відрізняються.

Реляційні бази даних з’явилися ще у 1970-х роках та швидко стали домінуючою технологією в галузі зберігання даних. Вони доказували свою ефективність та надійність протягом десятиліть. Існує багато реалізацій SQL БД, інструментів, документації та досвіду або “best-practices”, набутих за роки використання технології. Реляційні бази даних забезпечують зберігання у виді пов’язаних між собою таблиць даних. Ці таблиці мають фіксовану схему, використовують SQL (мову структурованих запитів) для керування даними та гарантують виконання ACID принципів під час роботи та виконання запитів.

Бази даних NoSQL відносяться до високопродуктивних нереляційних сховищ даних. Вони завоювали свою популярність простотою використання, чудовою підтримкою масштабованості та показниками стійкості (durability) та доступності (aviability). Замість об'єднання таблиць із нормалізованими даними NoSQL зберігає неструктуровані або напівструктуровані дані, часто у виді пар ключ-значення або в документах JSON. Такі бази даних, як правило, не надають гарантій ACID за межами одного розділу сховища, але на противагу цьому NoSQL забезпечує надшвидке виконання запитів, найчастіше всього менше секунди (див. діаграми на рис. 25-28), як для великих так і малих обсягів даних.

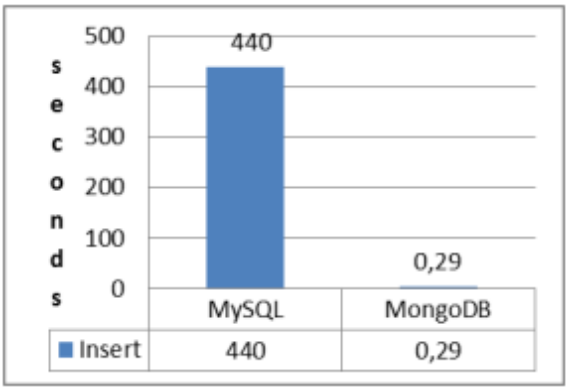


Рисунок 25. – Порівняння швидкості MySQL та MongoDB (вставка)

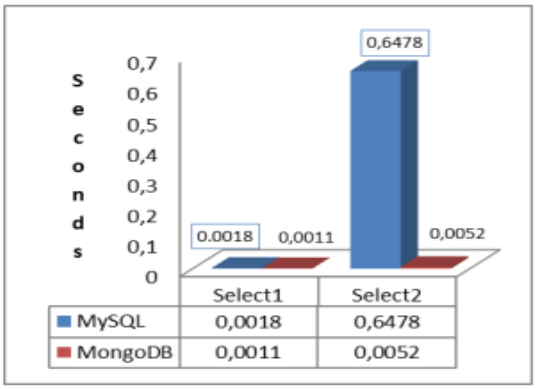


Рисунок 26. – Порівняння швидкості MySQL та MongoDB (читання)

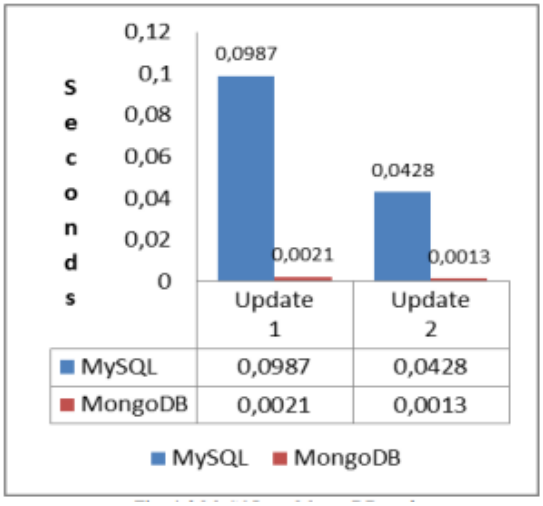


Рисунок 27. – Порівняння швидкості MySQL та MongoDB (оновлення)

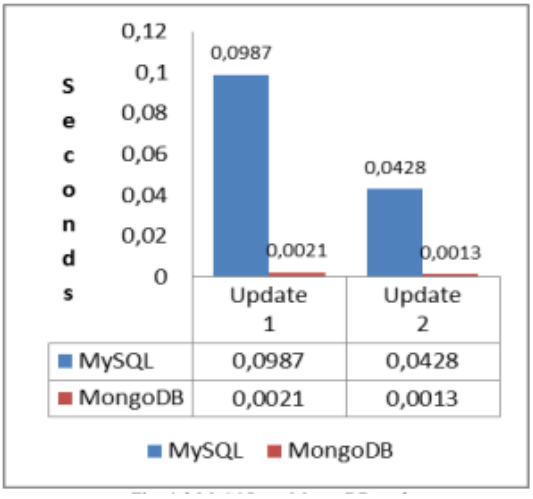


Рисунок 28. – Порівняння швидкості MySQL та MongoDB (видалення)

Враховуючи особливості роботи цих типів баз даних, можна зробити деякі висновки щодо доцільності використання SQL та NoSQL рішень для різних задач. Якщо потрібно зберігати дані у консистентному виді (із дотриманням правил щодо типів, значень та чітким збереженням відносин між даними), але при цьому не потребується надшвидкого доступу до цих даних у великих обсягах за короткий проміжок часу – наш вибір SQL. Проте, якщо на першому місці знаходиться саме можливість швидкого контролю над даними будь якого обсягу у будь який момент часу, а до самої схеми відсутні суворі вимоги, то NoSQL забезпечить найкраще рішення.

В AssistEasy в якості основного сховища даних використовується об’єктно-реляційна БД із відкритим вихідним кодом PostgreSQL, яка забезпечує надійне зберігання даних студентів та створених ними користувацьких груп. Особливу перевагу від використання PostgreSQL ми отримаємо під час операцій видалення, оскільки реляційне сховище забезпечить нам надійну підтримку транзакційності.

Оскільки веб-сервіс позиціонується як асистент, то в ньому використовується система запланованих повідомлень засобами telegram та email, а також автоматичне оновлення розкладу онлайн занять. Для забезпечення роботи такого сервісу, потрібно мати сховище для сповіщень та онлайн занять, яке зможе забезпечити швидкі CRUD операції над ними. Оскільки функціонал надсилання сповіщень та формування розкладу буде постійно (декілька разів на хвилину) виконувати polling (опитування) сховища на наявність нотифікацій для відсилання, а самі дані не потребують ніяких зв’язків між іншими даними системи, було обрано швидку документно орієнтовану NoSQL базу даних MongoDB. Окрім системи повідомлень, NoSQL рішення потребує й тип аутентифікації за допомогою ключа сесії, який використовує AssistEasy, оскільки при кожному запиті до серверу буде виконуватися пошук користувача у сховищі даних, що призводить до значного навантаження на сховище та потребує швидкої відповіді та високого показника доступності.

3.2.3. Брокер повідомлень RabbitMQ

Брокер повідомлень – це спеціальний компонент системи, який дозволяє додатками та сервісам обмінюватися даними між собою за допомогою черг та механізму видавець-споживач.

Черга є однією з форм структур даних, що схожа на чергу в реальному світі, де елементи розташовані один за одним та додати елемент можна тільки в кінець, а видалити тільки з початку.

Видавцем є компонент системи (додаток або сервіс), що створює дані, частіше всього події та команди, які передає брокеру повідомлень для подальшого їх споживання споживачем.

Споживачем є компонент системи (додаток або сервіс), що приймає ці дані та виконує над ними певні операції.

AsistEasy використовує RabbitMQ – популярний брокер повідомлень на основі стандарту AMQP із відкритим кодом. Він підтримує два базових шаблони доставки повідомлень:

1. **Обмін повідомленнями «від точки до точки».** Це шаблон розповсюдження, який використовується в чергах повідомлень із взаємозв’язком «один до одного» між відправником і одержувачем повідомлення. Кожне повідомлення в черзі надсилається лише одному одержувачу і споживається лише один раз. Обмін використовується, коли дії над повідомленням потрібно робити лише один раз. У цій системі і відправнику, і одержувачу потрібна гарантія того, що кожне повідомлення буде надіслано лише один раз.
2. **Публікація/підписка повідомлень**. В цьому шаблоні розповсюдження повідомлень, видавник кожного повідомлення публікує його в темі (topic), а кілька споживачів повідомлень підписуються на теми, з яких вони хочуть отримувати повідомлення. Усі повідомлення, опубліковані в темі, оброблюються багатьма підписниками. Це метод розповсюдження в стилі трансляції (broadcast), в якому між видавцем повідомлення та його споживачами існує зв’язок «один до багатьох».

Система використовує брокер для наступних дій:

- Для відправки повідомлень через email (див. схему на рис. 29)
- Для відправки повідомлень через telegram (див. схему на рис. 30)
- Для відправки користувацьких вебхуків (див. схему на рис. 31)

Рисунок 29. – Схема доставки повідомлень (email)

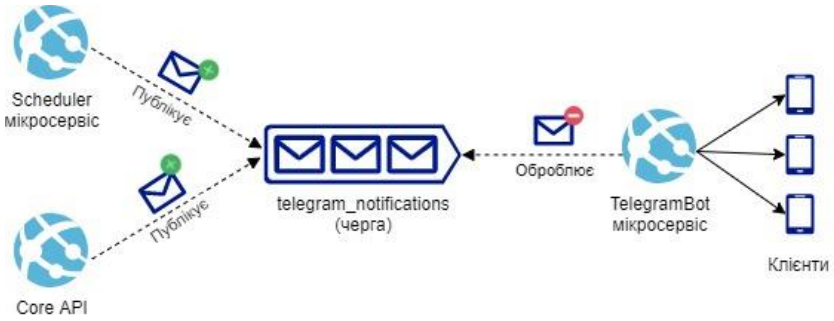


Рисунок 30. – Схема доставки повідомлень (telegram)

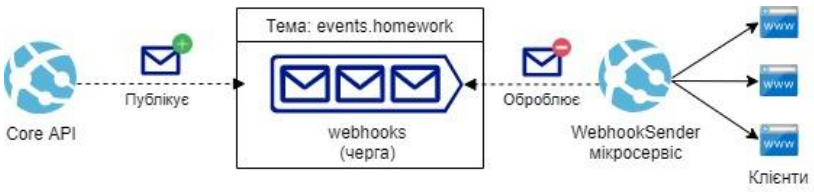


Рисунок 31. – Схема відправлення вебхуків

Використання брокеру повідомлень замість виклику ендпоінту іншого сервісу має наступні переваги:

- Програмні модулі системи та сервіси отримують слабке зчеплення

- Сервісу не потрібно очікувати завершення роботи іншого сервісу, що значно підвищує швидкість роботи та пропускну здатність та економить ресурси серверної машини
- Оскільки обробник повідомлень використовує чергу, він може оброблювати команди по мірі завантаженості, що робить систему більш надійною та витривалішою під час великої кількості запитів
- З’являється можливість одночасної публікації події для багатьох обробників, використовуючи лише один запит до брокеру

3.2.4. Azure BLOB storage

Для збереження та читання файлів вкладень та зображень, використовується сховище Azure Blob – хмарне рішення Microsoft для зберігання об’єктів. Сховище BLOB оптимізовано для зберігання величезної кількості неструктурованих даних. Неструктуровані дані – це дані, які не відповідають певній моделі даних або визначенню, наприклад текстові або двійкові дані. Дані зберігаються у контейнерах (див. рис. 32-33). Рішення Azure дозволяє:

- Відображати та завантажувати збережені файли прямо в браузері за допомогою SAS посилання
- Керувати рівнем доступу до файлів
- Стрімити відео та аудіо напряму зі сховища
- Підтримує логування та відновлення видалених файлів
- Легко завантажувати та керувати файлами за допомогою бібліотеки Azure.Storage.Blobs для мови програмування C#

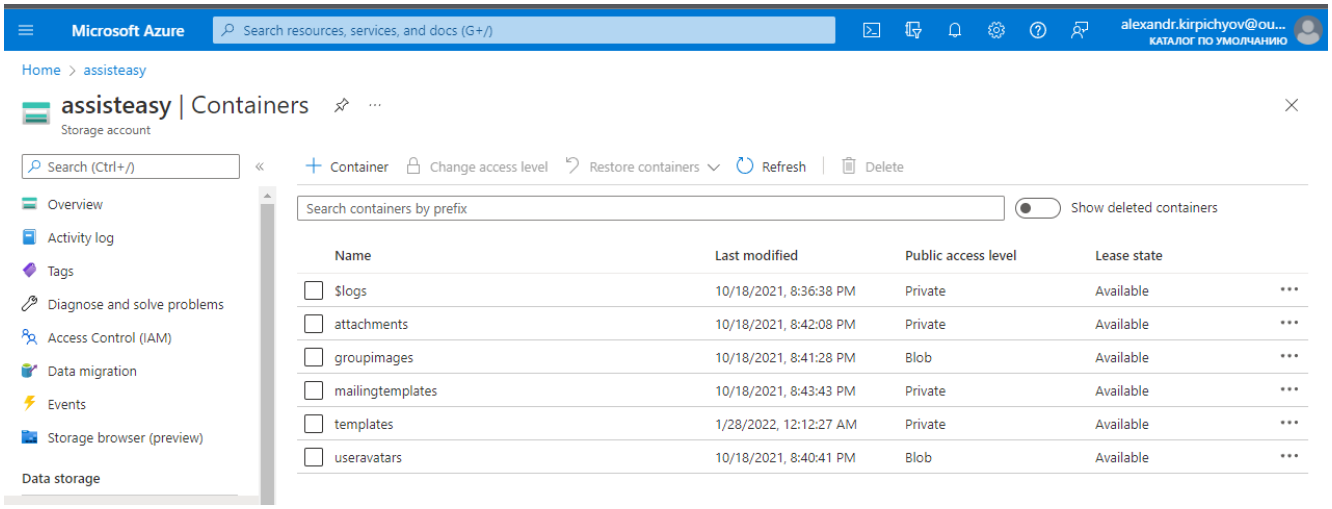


Рисунок 32. – Контейнери для файлів

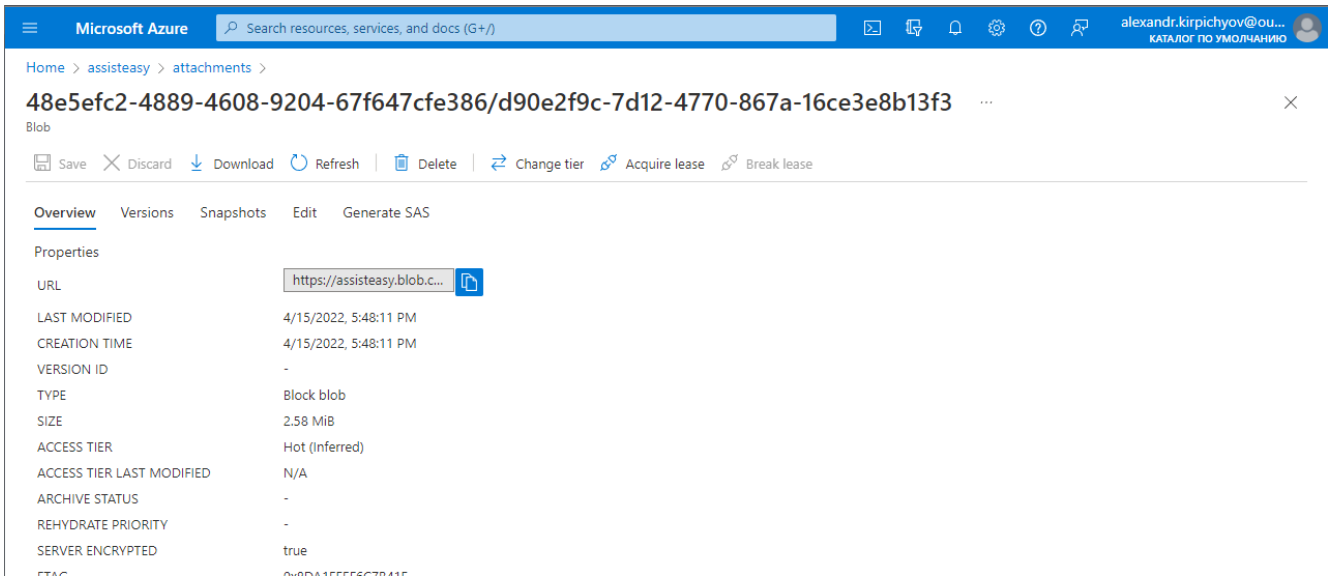


Рисунок 33. – Інформація про збережений файл

3.2.5. Шарова структура

Мікросервіси всередині поділені на шари (див. рис. 34), що робить код значно простішим для розуміння, а значить і для модифікації та розробки, оскільки модулі системи логічно розділені (див. табл. 6).

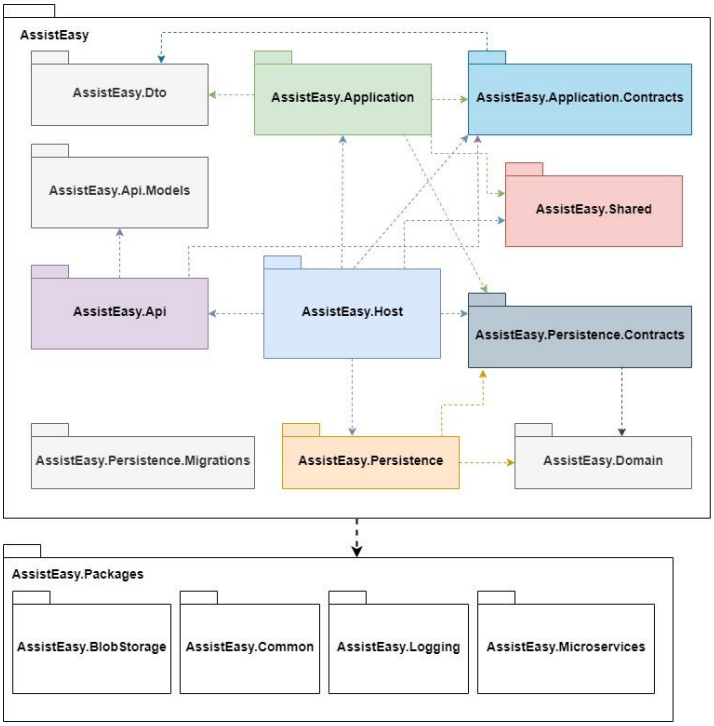


Рисунок 34. – Діаграма пакетів core додатку

Таблиця 6. – Перелік та призначення пакетів

Пакет	Призначення
AssistEasy	Батьківський пакет, що містить весь функціонал core сервісу
AssistEasy.Host	Містить в собі програмні компоненти для конфігурування, запуску та хостингу веб додатку
AssistEasy.Api	Містить в собі контролери, валідатори запитів та правила мапінгу для моделей запиту
AssistEasy.Domain	Містить entity моделі, що представляють записи в БД
AssistEasy.Dto	Містить моделі, що використовуються шаром Application
AssistEasy.Api.Models	Містить моделі, що використовуються шаром Api
AssistEasy.Persistence (DataAccess)	Містить логіку підключення до джерела даних та обробки даних з неї (одиниця роботи та репозиторії)
AssistEasy.Persistence.Contracts	Містить інтерфейси (контракти) для роботи з джерелом даних

Пакет	Призначення
AssistEasy.Persistence.Migrations	Містить спеціальний консольний додаток для створення та застосування міграцій для БД
AssistEasy.Application (BusinessLogic)	Містить у собі сервіси та основну логіку додатку. Також містить правила мапінгу для data transfer object моделей.
AssistEasy.Application.Contracts	Містить інтерфейси (контракти) для роботи з основною логікою додатку
AssistEasy.Shared	Містить у собі програмні модулі та розширення, які є спільними для декількох шарів
AssistEasy.BlobStorage	Містить у собі програмні модулі для роботи з Azure BLOB storage
AssistEasy.Common	Містить у собі найчастіше використовувані програмні компоненти та розширення
AssistEasy.Logging	Містить у собі розширення для конфігурації логування
AssistEasy.Microservices	Містить у собі контракти мікросервісів для взаємодії між собою та брокером повідомлень

Оскільки шари використовують різні моделі для взаємодії один з одним, то виникає потреба у використанні механізму мапингу. Мапінг – це процес перетворення даних з одного типу в інший. Може використовуватися, як ручний, так і автоматичний мапінг. AssistEasy використовує автоматичний мапінг на основі прописаних правил за допомогою бібліотеки Automapper (див. рис. А. 1-2).

3.2.6. Використання принципів ООП

Архітектура проекту передбачає активне слідування важливим принципам ООП, таких як SOLID, DRY, KISS та YAGNI. Їх використання дозволяє писати код, який буде легше модифікувати, уникати повторень та може зменшити час, потрібний для розробки програмних модулів.

YAGNI (You aren’t gonna need it). Цей принцип про те, що потрібно писати лише той код, який нам точно потрібен зараз або знадобиться для наступних змін. Не треба писати код або залишати старий код “на майбутнє, бо може згодиться”.

KISS (Keep it stupid simple). Цей принцип про те, що не потрібно занадто сильно ускладнювати код, особливо там де можна використати просте рішення.

DRY (Don’t repeat yourself). Базовий принцип про те, що не треба повторюватися. Замість цього потрібно виносити повторюваний код у методи, класи, модулі та бібліотеки. Наприклад, AssistEasy має цілий проект з винесеними пакетами (AssistEasy.Packages), що містять класи та методи, які найчастіше використовуються (див. рис. 35-36).

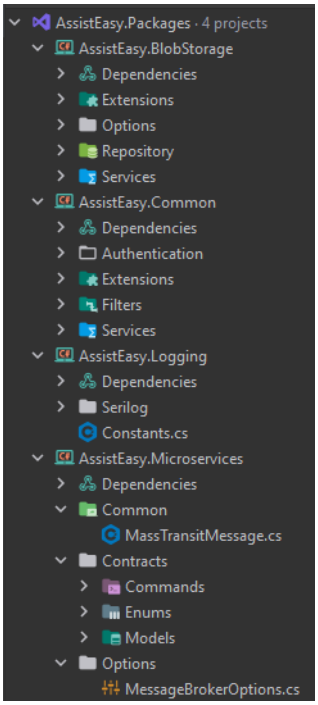


Рисунок 35. – Проект з винесеними пакетами

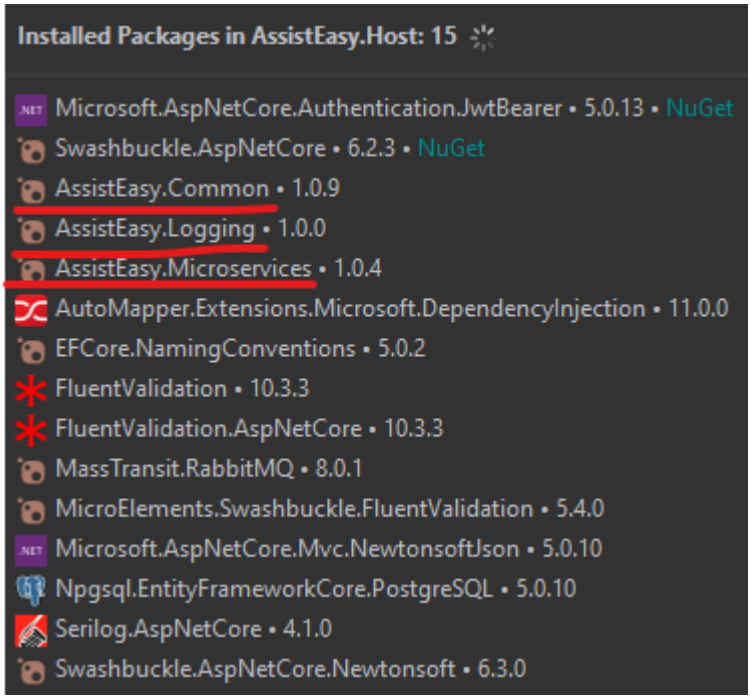


Рисунок 36. – Приклад використання пакетів в core додатку

SOLID. Абревіатура, що складається із принципів:

- S (Single responsibility) – кожен компонент системи повинен мати лише одну відповідальність. Приклади використання в коді проекту наведено нижче (див. рис. А. 3-4).
- O (Open close) – кожен модуль системи повинен бути відкритий до розширення, але закритий для модифікації.

- L (Liskov substitution) – об’єкти батьківського типу можуть бути заміщені об’єктами дочірнього типу. Цей принцип підтримується мовою C# автоматично.
- I (Interface segregation) – інтерфейси повинні бути розділені та містити тільки ті контракти, які відносяться до контракту. Приклади використання в коді проекту наведено нижче (див. рис. А. 5-7).
- D (Dependency inversion) – програмні компоненти повинні залежати від інтерфейсів, а не реалізації. Приклади використання в коді проекту наведено нижче (див. рис. А. 8-9).

3.3. Моделювання таблиць та підключення до баз даних

3.3.1. Використання ORM

Веб-сервіс використовує ORM для підключення до баз даних PostgreSQL та MongoDB. ORM (Object-relational mapping) – це технологія, яка дозволяє взаємодіяти із базою даних за допомогою модулів та об’єктів мови програмування. Усі таблиці та записи в джерелі даних представлені класами та об’єктами, а для виконання CRUD (Create Read Update Delete) операцій викликаються відповідні методи. ORM використовує провайдери або драйвери баз даних для транслювання запитів до конкретної БД. Технологія може мати свою особливості щодо транслювання запитів та створення схеми бази – це залежить від бібліотеки та мови програмування.

Для підключення до PostgreSQL використовується ORM Entity Framework Core та провайдер Npgsql. Це потужна бібліотека, яка вміє транслювати код LINQ (Language-Integrated Query) в запит до БД. Також є система трекінгу змін, яка дозволяє просто оновлювати властивості об’єкту та викликати метод для збереження, що виглядає як звичайна операція ООП. Приклади використання наведено нижче (див. рис. А. 10-12).

Для підключення до MongoDB використовується ORM MongoDB.Driver. Ця бібліотека є швидкою та легкою. Таблиці тут також представлені у виді колекцій,

але немає трекінгу, а замість контексту використовується ініціалізація підключення до кожної колекцій (тому бібліотеку було обернено у самописний контекст). Запити до MongoDB можна писати як за допомогою LINQ, так і спеціальних об’єктів будівельників. Приклади використання наведено нижче (див. рис. А. 13-14).

3.3.2. Моделювання таблиць PostgreSQL

Схему бази даних представлено через ERD (Entity-Relationship diagram), яка наглядно показує сутності та зв’язки між ними (див. рис. 37).

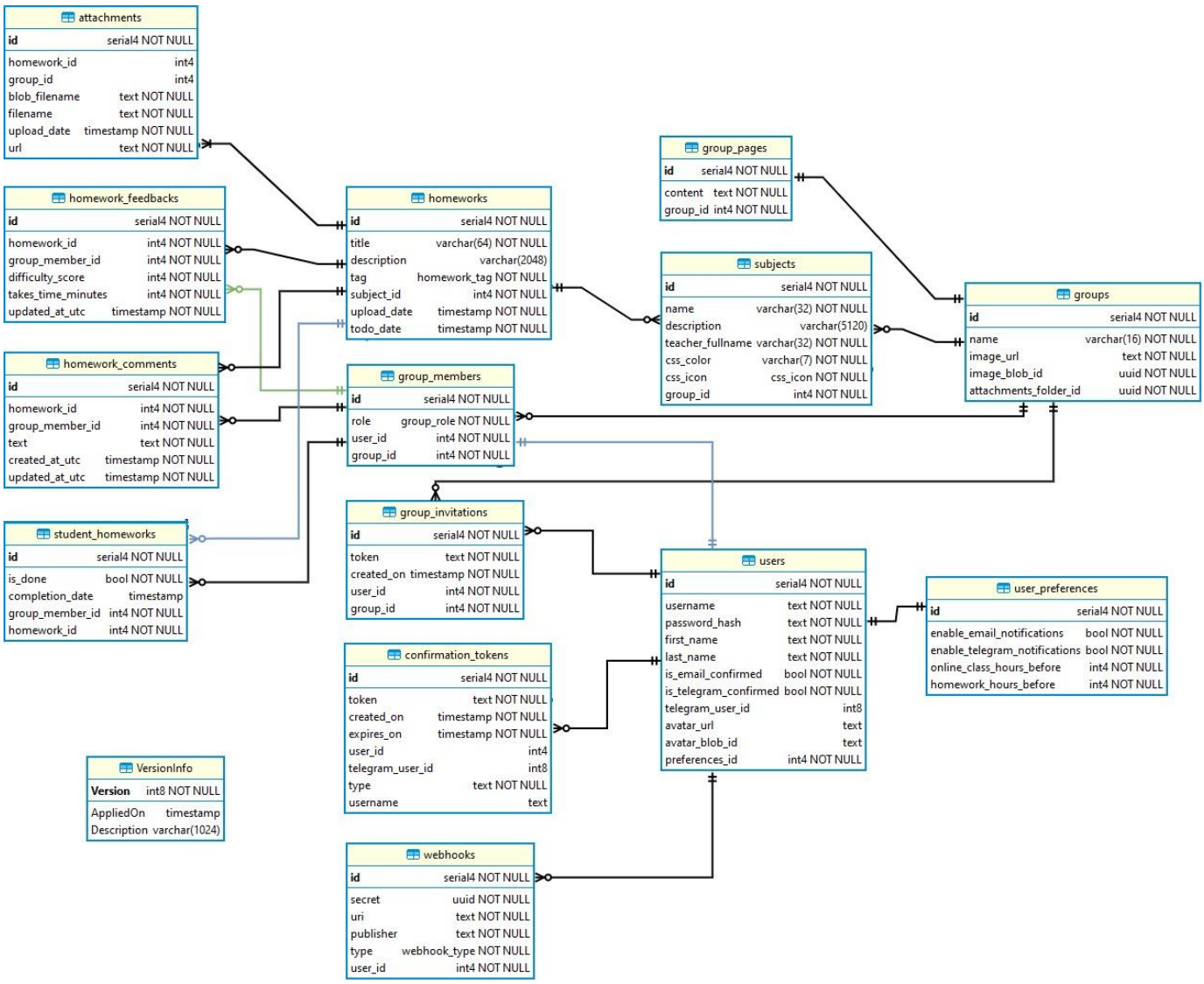


Рисунок 37. – ER діаграма для БД PostgreSQL

3.3.3. Моделювання таблиць MongoDB

Оскільки взаємодія з БД буде здійснюватися за допомогою об’єктів, а MongoDB не потребує схеми, то моделювання таблиць здійснено за допомогою діаграми класів (див. рис. 38).

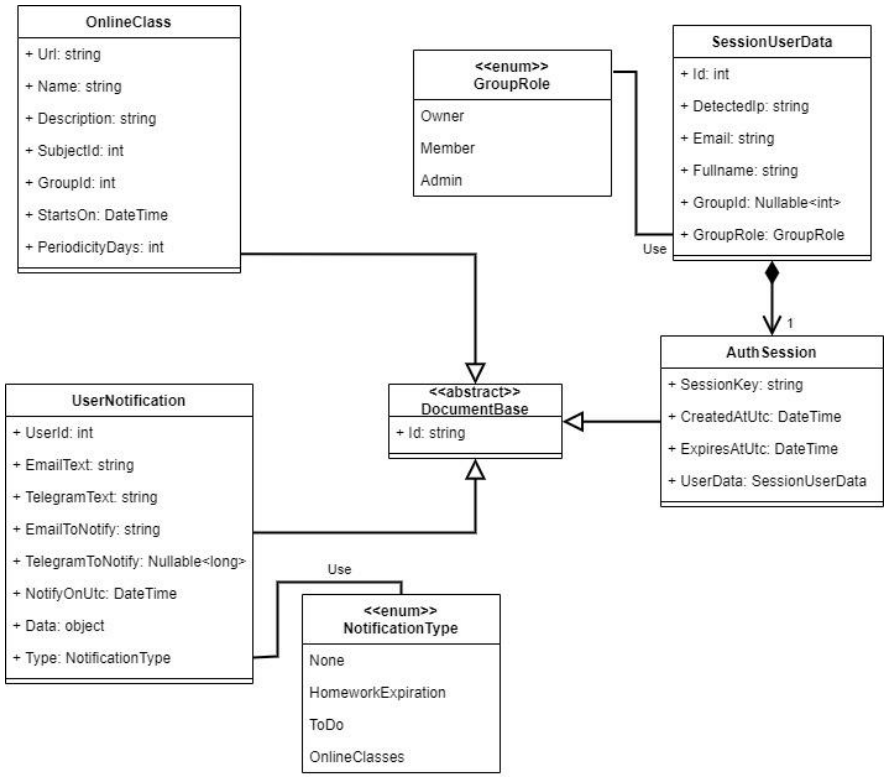


Рисунок 38. – Діаграма класів для сутностей MongoDB

3.3.4. Використання міграцій

MongoDB зазвичай не потребує створення міграцій через те, що схема там є динамічною та колекції створюються самі при першій вставці. PostgreSQL в свою чергу є об'єктно-реляційною системою тому потребує створення схеми. Оскільки використовується ORM Entity Framework Core, то є декілька підходів: Code First, Database First та Model First. AssistEasy використовує підхід Database First, оскільки це дозволяє краще контролювати схему, що використовується.

Для створення та застосування міграцій використовується консольний додаток на основі бібліотеки Fluent Migration, який дозволяє описувати схему міграцій за допомогою мови C#, використовуючи зрозумілий біглий (fluent)

інтерфейс (див. рис. А. 15). Для застосування створених міграцій потрібно тільки вказати строку підключення до БД та запустити додаток.

3.4. Автентифікація

AssistEasy використовує декілька типів автентифікації. Механізм на основі сесійного ключа використовується для виконання запитів до API для користувачів, а на основі API ключа – для інших мікросервісів, зокрема для планувальнику (Scheduler мікросервіс).

3.4.1. Сесійний ключ

Автентифікація на основі сесійного ключа використовує механізм, при якому користувач робить запит, що містить необхідні дані для входу (email та пароль), а у відповідь отримує спеціальний ключ, використовуючи який може робити запити до системи (див. рис. 39). Сервіс в свою чергу буде шукати сесійний запис, створений у БД по цьому ключу. Якщо такий існує, буде проініцілізовано контекст користувача та почнеться подальше обробка запиту, інакше – відповідь з кодом 401.

Такий підхід на відміну від JWT токену, дозволяє миттєво застосувати зміни до сесії користувача (наприклад, при оновлені ролі чи інших атрибутів доступу), а також дозволяє забрати доступ до системи чи інвалідувати сесію практично без жодних затримок. Ця особливість досягається завдяки тому, що ключ не містить у собі ніяких даних і є лише ідентифікатором, за яким відбувається пошук у БД, де містяться необхідні дані облікового запиту (див. рис. 40). Звідси випливає інша особливість – треба постійно ходити в БД при кожному запиті, але ця проблема легко вирішується використанням надшвидкої NoSQL бази.

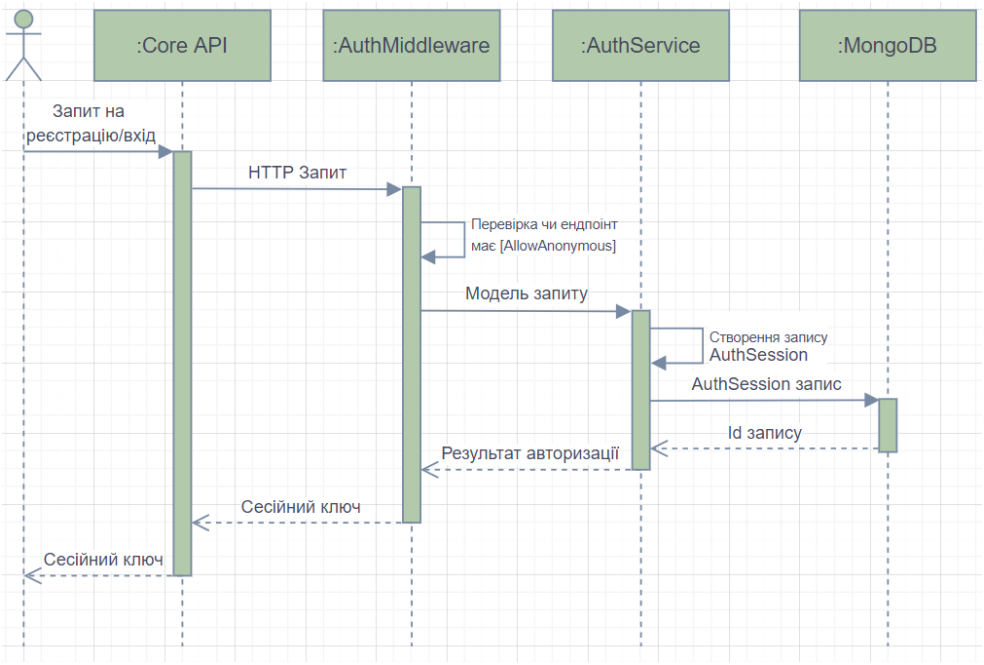


Рисунок 39. – Діаграма послідовності - вхід/реєстрація (сесійний ключ)

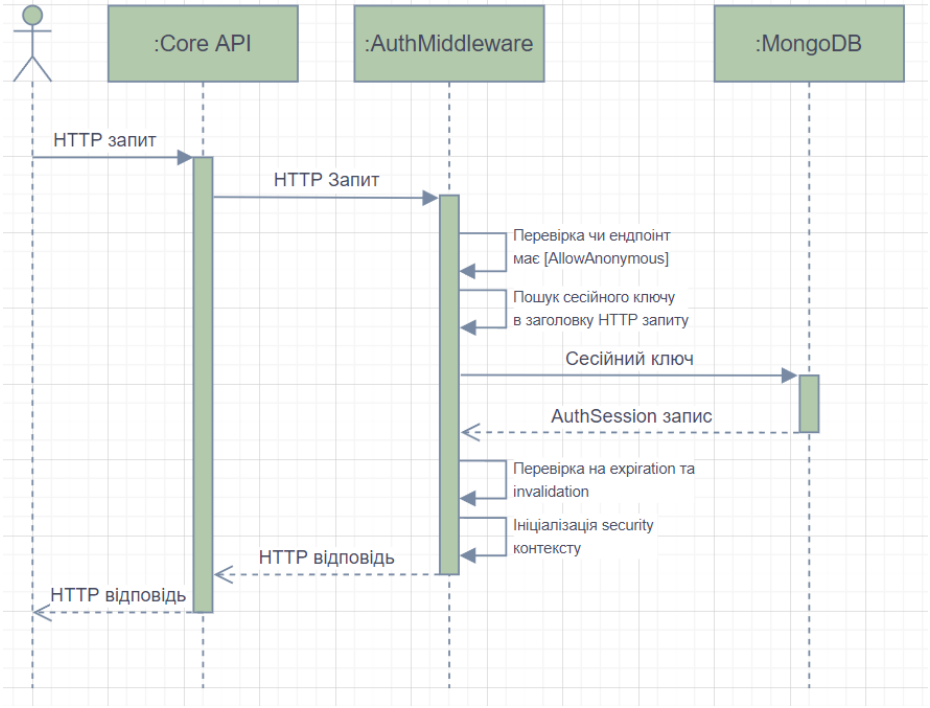


Рисунок 40. – Діаграма послідовності - виконання запиту (сесійний ключ)

3.4.2. API ключ

Автентифікація на основі API ключа використовує механізм, при якому користувач робить запит, що містить спеціальний ключ, створений для комунікацій з сервісом. Якщо ключ співпадає з ключем, вказаним у сервісі, що викликається, то

дозволяється подальше виконання запиту, інакше повертається код 401 (див. рис. 41). Такий простий спосіб підходить при міжмікросервесній комунікації, коли не потрібно передавати будь яку інформацію, але треба дозволити доступ лише авторизованим користувачам.

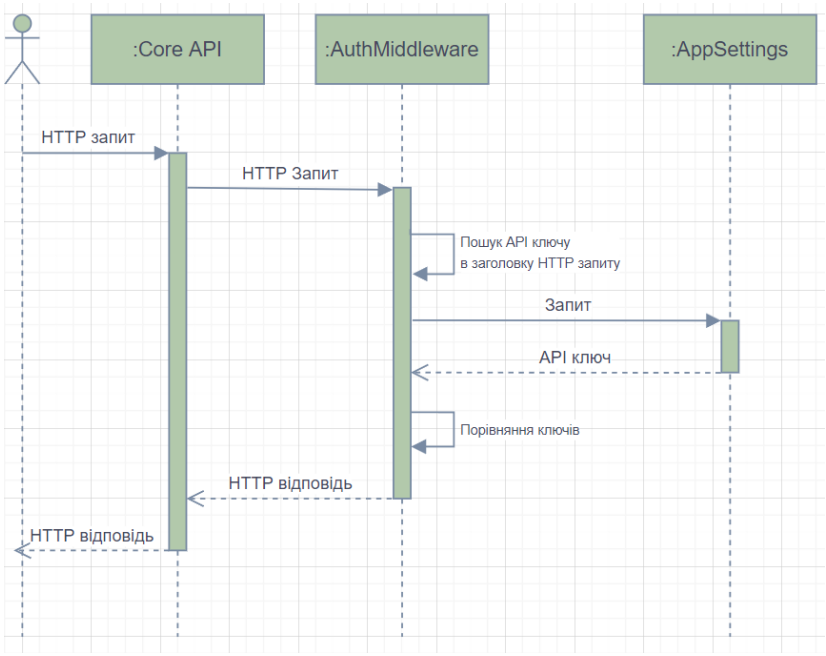


Рисунок 41. – Діаграма послідовності - виконання запиту (API ключ)

3.5. Система сповіщень

У веб-сервісі реалізовано систему миттєвих та запланованих сповіщень засобами email та telegram. Для відправлення електронних листів використовується бібліотека MailKit, а для відправлення повідомлень через telegram – бібліотека Telegram.Bot.

Миттєві сповіщення додаються до черги для обробки іншим мікросервісом, що відповідає за розсилання, одразу після створення (див. рис. 42). Також до цієї категорії можна віднести вебхуки, які одразу додаються в чергу для викликання іншим мікросервісом (WebhookSender) при виникненні події. А заплановані сповіщення мають певну дату та час (у форматі UTC), коли їх треба додати в чергу

для розсилання. Такі нотифікації після створення додаються в колекцію сповіщень в MongoDB, яку в подальшому мікросервіс планувальник (Scheduler) за допомогою Job-ів та полінгу (періодичного опиту) додає в чергу.

Job – це така програма реалізаця, що працює на фоні на періодично (залежно від встановленого часу) виконує певні функції. AssistEasy має дві Job-и, одну для полінгу нотифікацій, а другу для оновлення розкладу онлайн занять. Для реалізації даної технології використовується бібліотека Quartz.Net.

Полінг – це такий програмний алгоритм, при якому періодично викликається запит до якогось ресурсу (наприклад БД або API). Коли викликається відповідна job-а (кожну хвилину), сервіс робить запит до колекції сповіщень в MongoDB, щоб отримати ті сповіщення, які вже потрібно відправити. Якщо такі знайдено, то вони додаються в чергу на відправлення та видаляються з колекції (див. рис. 43).

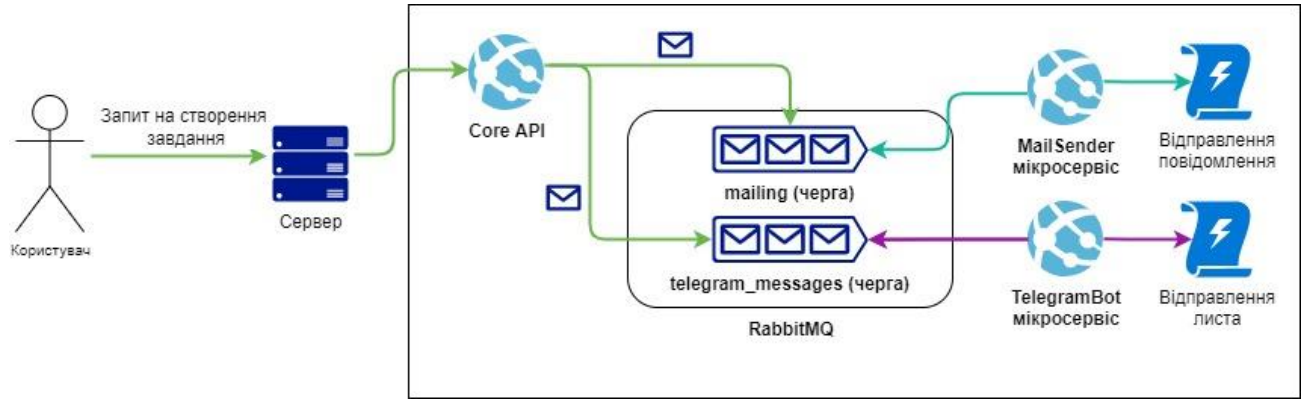


Рисунок 42. – Схема відправлення миттєвих сповіщень

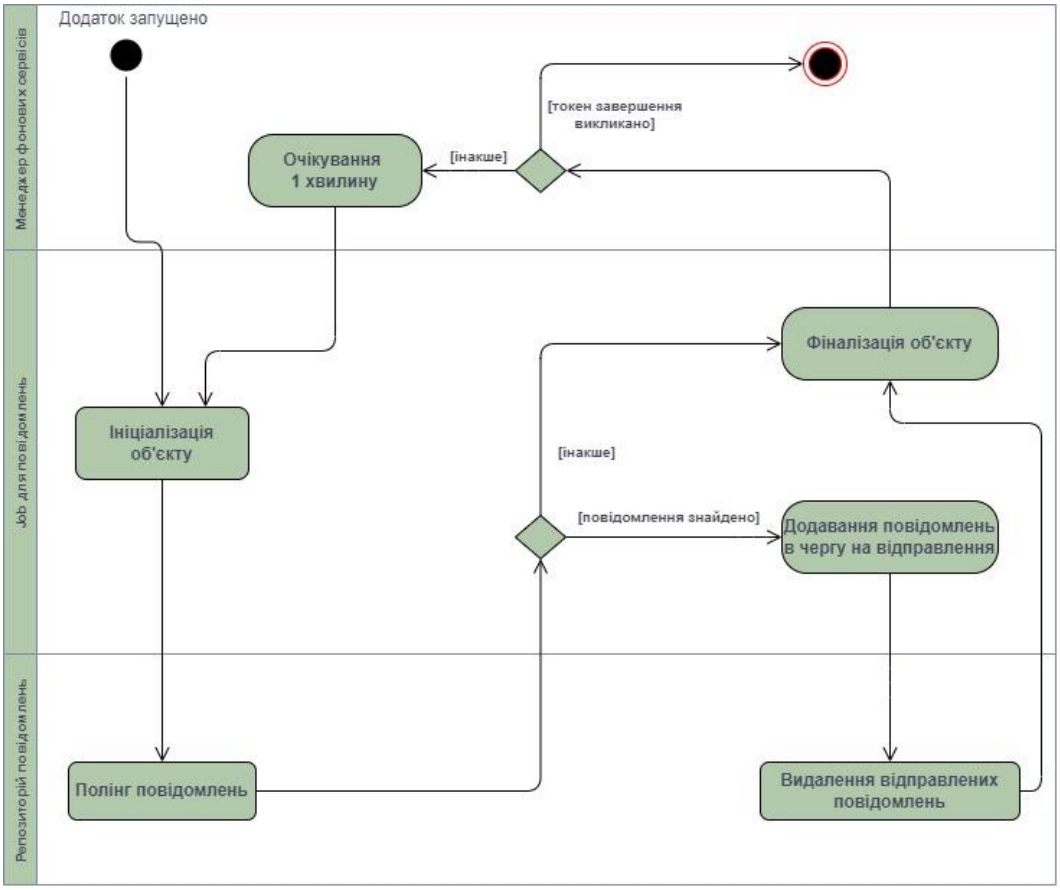


Рисунок 43. – Діаграма діяльності (відправлення повідомлень планувальником)

3.6. Використання шаблонів проектування

Важливою частиною розробки програмного забезпечення є використання шаблонів проектування. Шаблон проектування – це загальне рішення для поширеної проблеми в певному контексті розробки програмного забезпечення, яке може бути використано безліч разів. Вони поділяються на структурні, твірні та шаблони поведінки.

3.6.1. IoC та Dependency injection

Інверсія керування та ін’єкція залежностей використовується для зменшення зачеплення та рівню залежностей між компонентами програмного забезпечення. Цей шаблон є невід’ємною частиною літери “D” в принципах SOLID. Суть шаблону полягає в тому, щоб використовувати інтерфейси замість реалізацій, а для виклику реалізації використовувати контейнер, для якого заздалегідь створюється

конфігурація, де вказано яку реалізацію використовувати для певного інтерфейсу або типу. При створенні певного типу використовується рекурсивне сканування на конструктор та його параметри, оскільки створюваний об’єкт може мати інші залежності, які також повинні бути зареєстровані в контейнері. Веб-сервіс використовує розроблену компанією Microsoft вбудовану бібліотеку для реалізації цього патерну. Приклади використання в проекті наведено нижче (див. рис. А. 16-17).

3.6.2. Repository та Unit of Work

Репозиторій та Одиниця роботи це два шаблони, які найчастіше за все використовуються разом.

Репозиторій використовується для винесення логіки роботи з джерелом даних в окрему абстракцію, що дозволяє використовувати більш слабкі зв’язки між компонентами та мати можливість зміни джерела даних без внесення змін у методи, що викликають репозиторій, а також спрощує написання unit тестів.

Одиниця роботи використовується для інкапсуляції усієї логіки для роботи з джерелом даних та контролю транзакцій, що дозволяє використовувати один контекст бази даних для всіх репозиторіїв та застосовувати зміни в межах певної транзакції. Приклади використання шаблонів в проекті наведено нижче (див. рис. А. 18-19).

3.6.3. Chain of responsibility

Ланцюжок обов’язків – це поведінковий шаблон, що дозволяє виконувати обробку запитів послідовно через всіх зареєстрованих обробників. Кожен обробник може обробити запит та зупинити ланцюжок або передати запит далі по ланцюжку. Братом близнюком цього патерну проектування є архітектурний шаблон Pipeline.

Цей шаблон активно використовується в AssistEasy оскільки фреймворк ASP.NET використовує його для обробки HTTP запитів. HTTP протокол передбачає запит, що має певну структуру зображену на схемі (див. рис. 44). Під час процесінгу запиту, кожен обробник, який в ASP.NET називається Middleware,

може зчитувати HTTP запит та виконувати над ним дії (див. рис. 45), а також він може змінювати його, записуючи туди необхідну інформацію, таку як: код відповіді, тіло та заголовки.

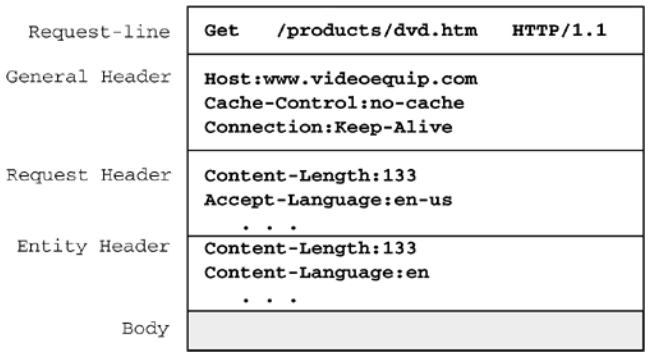


Рисунок 44. – Структура HTTP запиту

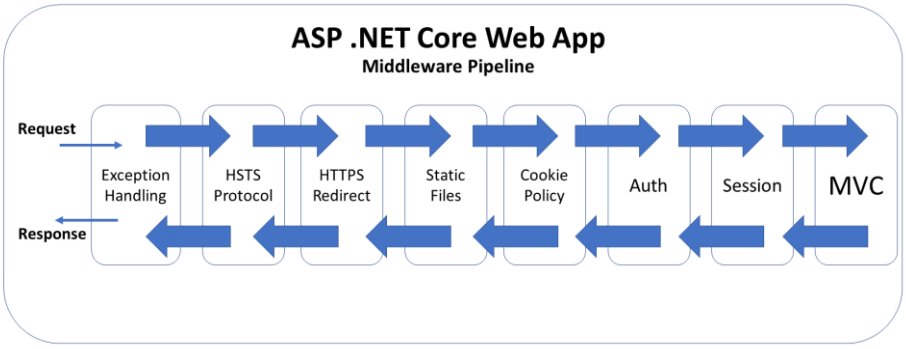


Рисунок 45. – Приклад ланцюгу обробки HTTP запиту в ASP.NET

Для реалізація автентифікації було розроблено middleware для обробки запитів (див. рис. А. 20), яке перевіряє запит на наявність заголовку з ключем сесії та вимоги ендпоінту до авторизації. Якщо ендпоінт потребує авторизації та хедер містить ключ, то відбувається пошук запису сесії в базі та ініціалізація контексту користувача.

Також API використовує механізм для мапингу виключень на відповідний статус код та тіло запросу, що містить помилки та необхідну інформацію (див. рис.

А. 21). Цей механізм реалізовано за допомогою action фільтру, який також є частиною middleware.

3.6.4. Model View Controller

Model View Controller є найпопулярнішим шаблоном проектування для створення веб додатків. Він дозволяє чітко розділити обов’язки між абстракціями та описує зв’язки між ними (див. рис. 46). Фреймворк має повну підтримку цього шаблону у проекті за замовчуванням, де для Web API відображенням (View) можна вважати JSON який повертає контролер. Тому AssistEasy активно вискористовує компоненти MVC (див. рис. А. 22-24).

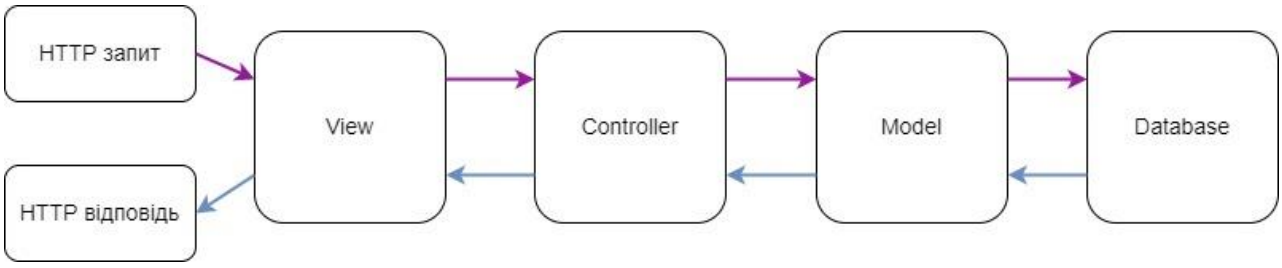


Рисунок 46. – Шаблон проектування MVC в ASP.NET Core Web API

3.6.5. Builder

Будівельник – це твірний шаблон, що дозволяє зручно створювати об’єкти на основі виклику спеціальних методів, що будуть конфігурувати його та повертати один й теж самий об’єкт будівельника, поки не буде викликано Build метод, який збудує необхідний компонент. Фреймворк використовує його для конфігурування та створення веб додатку (див. рис. А. 25-26). Також для мікросервісу AssistEasy.MailSender розроблено будівельник email повідомлення (див. рис. А. 27-28).

3.7. Swagger UI

Важливою частиною розробки відкритого прикладного програмного інтерфейсу є документація до ендпоінтів, запитів та відповідей. Контракт API можна побудувати вручну або використовуючи засоби автодокументації, одним з останніх є набір бібліотек та інструментів Swagger. Після налаштування в системі та використання спеціальних атрибутів (див. рис. А. 29-30), при запуску сервісу буде генеруватися специфікація OpenAPI. Окрім візуального представлення контракту, інструмент дозволяє одразу спробувати API за допомогою інтерактивного інтерфейсу Swagger UI (див. рис. 47-49).

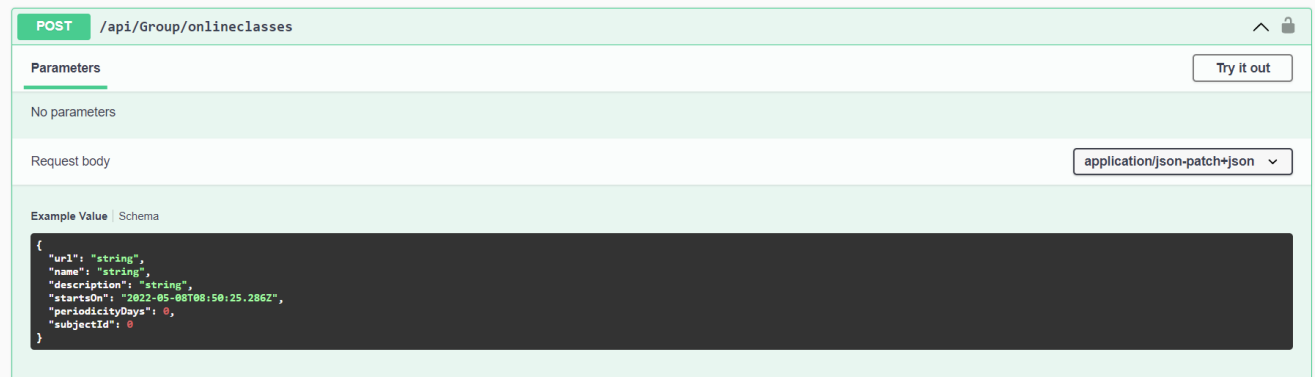


Рисунок 47. – Специфікація API згенерована Swagger

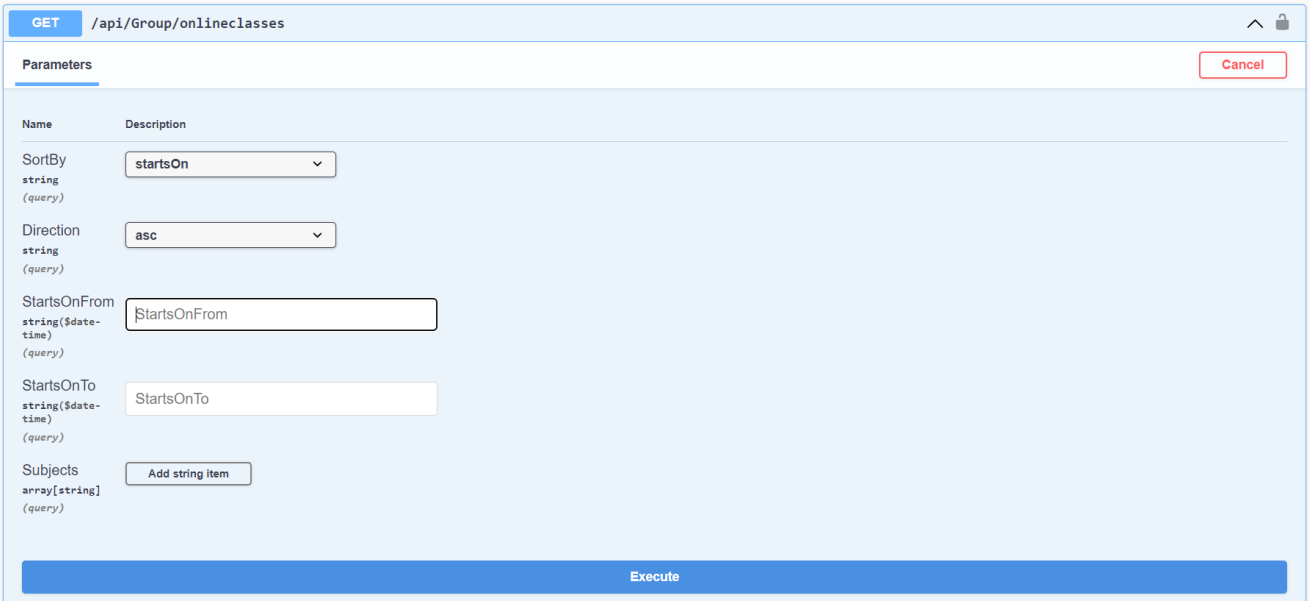


Рисунок 48. – Специфікація API згенерована Swagger

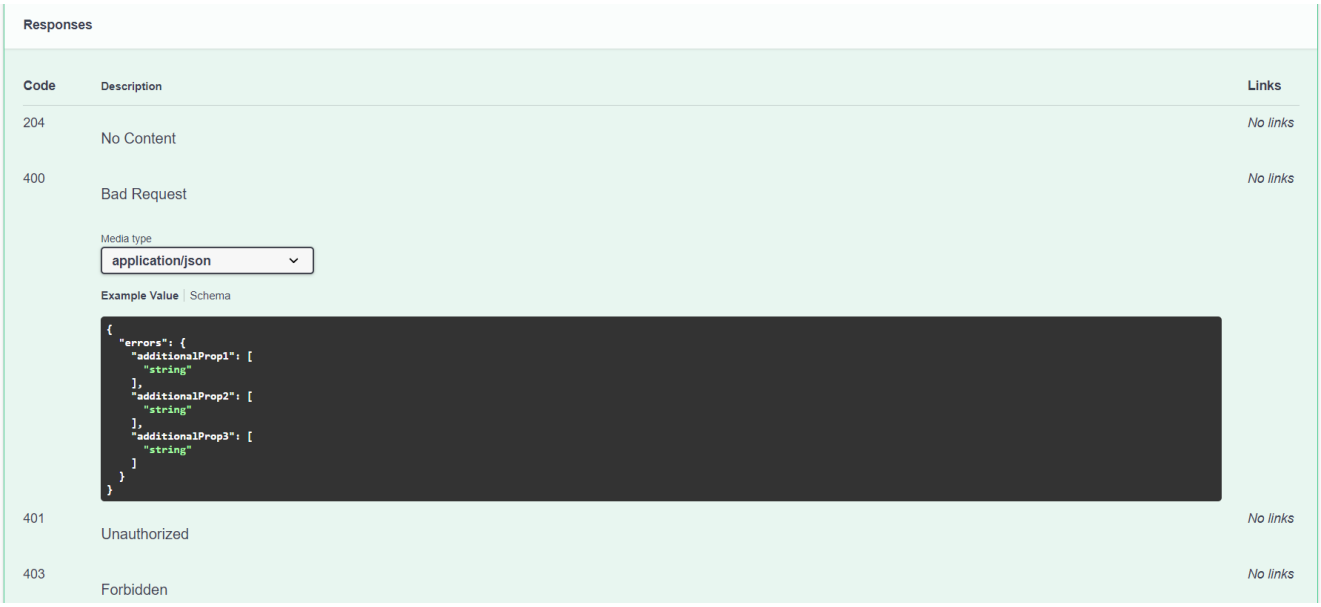


Рисунок 49. – Специфікація API згенерована Swagger

3.8. React JS

Для демонстрації можливостей прикладного програмного інтерфейсу сервісу AssistEasy було розроблено веб-клієнт за допомогою технології React JS.

React JS – це безкоштовна JavaScript бібліотека з відкритим вихідним кодом для створення інтерфейсів користувача, що складаються з компонентів. Розробка активно підтримується компанією Meta (стара назва – Facebook) та розробниками з усього світу. Бібліотека дозволяє легко й швидко створювати SPA (Single Page Application) додатки за допомогою мови програмування JavaScript. В основі технології лежить система керування станами та рендерінгом DOM (Document Object Model). React використовує віртуальний DOM, що дозволяє надзвичайно швидко відображати зміни на UI.

4. ТЕСТУВАННЯ СИСТЕМИ

4.1. Написання Unit тестів

Оскільки однією з вимог до розроблюваної системи є покриття основних модулів системи Unit тестами, було використано потужний та швидкий фреймворк з відкритим вихідним кодом - xUnit.Net. Також для аналізу покриття використовується інструмент dotCover (див. рис. 50-51), що є частиною IDE Rider при використанні ліцензії DotNet Ultimate.

Підхід до написання та найменування методів тестів відповідає AAA (Arrange, Act, Assert). Під час виконання Arrange (організаційної) частини відбувається налаштування усіх необхідних об’єктів, моків, стабів, фейків та системи, що тестується (SUT – System Under Testing). Act (від англ. – дія) частина викликає відповідні частини тестованого модулю. Assert (від англ. – стверджувати) частина використовується для задання перевірок при виконанні яких тест зараховується як успішно пройдений. Приклад unit тестів в системі наведено нижче (див. рис. А. 31-32).

В тестах активно використовуються моки. Mock – це спеціальний об’єкт, який дозволяє налаштовувати очікувану логіку та результат виконання методів окремих програмних компонентів в залежності від даних та тестового сценарію. Окрім цього, вони дозволяють перевірити чи викликалися певні методи взагалі, скільки разів, в якому порядку та з якими параметрами. Це дозволяє легко провести тестування модулю за допомогою налаштування відповідей інших модулів, на які має залежності тестований програмний компонент. Для створення моків використовується бібліотека FakeItEasy.

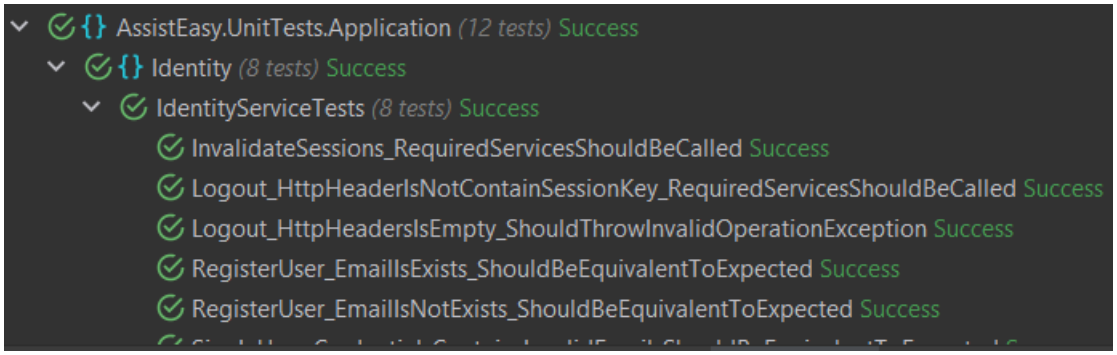


Рисунок 50. – Результати виконання тестів для IdentityService

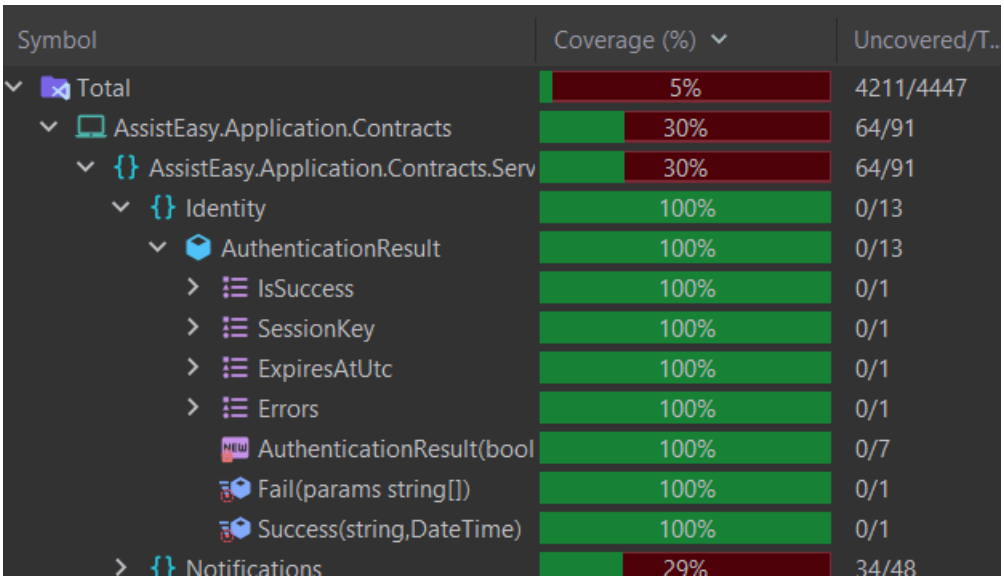


Рисунок 51. – Аналіз покриття програмних модулів

4.2. Ручне тестування

Для перевірки роботи веб-сервісу було обрано ручне тестування, оскільки цей спосіб є самим простим та не потребує написання автоматизованих тестів, як у випадку з автоматизованим тестування. Первинне тестування виконує розробник під час створення функцій програмного продукту, але для додаткової перевірки основних функцій системи було проведено SMOKE тестування, що дозволяє провести поверхневу перевірку головного функціоналу та виявити, чи придатна програма до використання взагалі. Для виконання тестування було складено тест кейси, оскільки це дозволяє чітко сформувати основі моменти для тестування, що включають в себе передумови, кроки, вхідні дані та очікуваний результат.

Таблиця 7 – Тест кейс 1 Реєстрація нового користувача

Тест кейс 1. Реєстрація нового користувача.	
Передумови	E-mail не зареєстровано в системі
Кроки	1. Перейти на сторінку реєстрації 2. Ввести значення в поля форми 3. Натиснути на кнопку “Зареєструватися”
Дані на вхід	Email = ‘alexandr.kirpichyov@gmail.com’ Ім’я = ‘Олександр’ Прізвище = ‘Кирпичов’’ Пароль = ‘Qwerty7@’ Повторіть пароль = ‘Qwerty7@’
Очікуваний результат	• Перехід на головну сторінку профілю користувача • Користувач з’явився в системі та БД • Користувач отримав лист із підтвердженням пошти

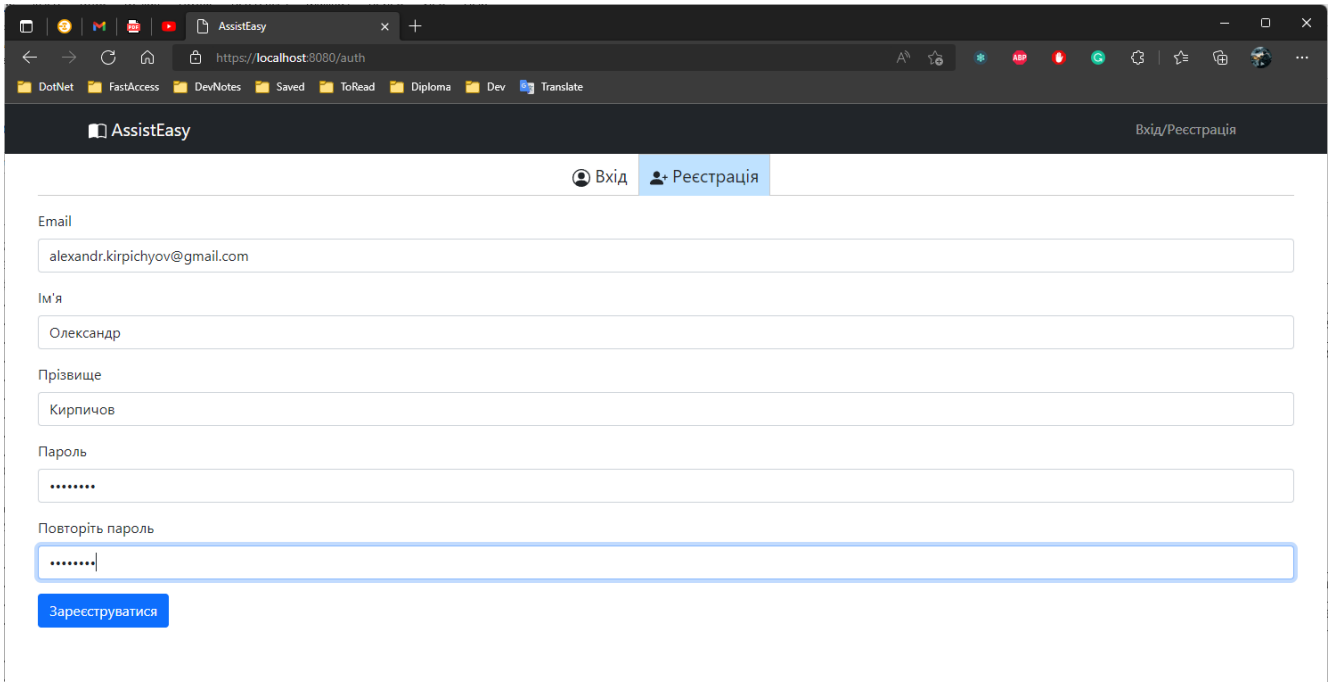


Рисунок 52. – Сторінка реєстрації AssistEasy [Тест кейс 1]

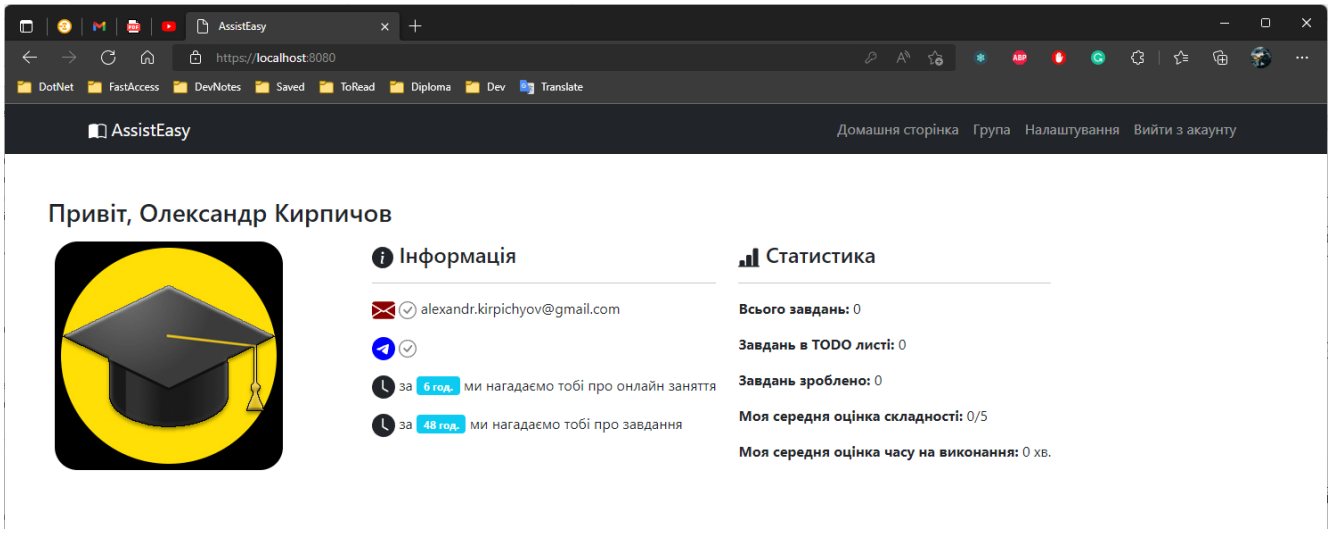


Рисунок 53. – Сторінка профілю AssistEasy [Тест кейс 1]

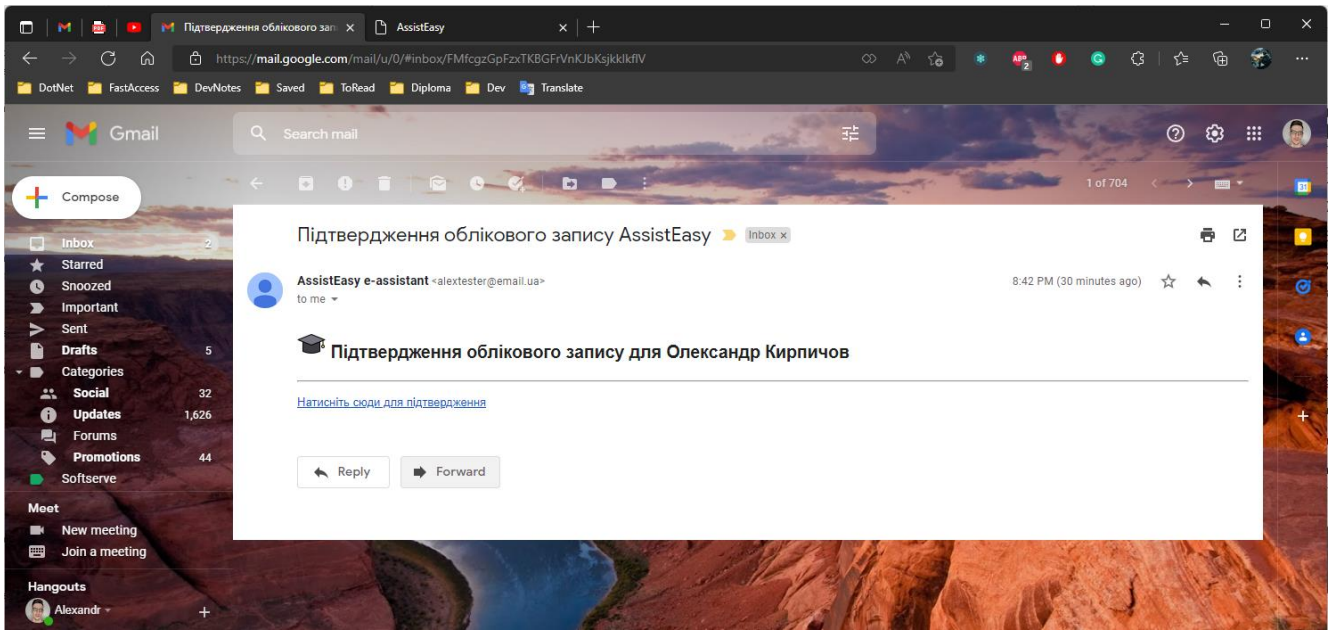


Рисунок 54. – Лист із підтвердженням облікового запису AssistEasy [Тест кейс 1]

Таблиця 8 – Тест кейс 2. Вхід для існуючого користувача (правильні дані)

Тест кейс 2. Вхід для існуючого користувача (правильні дані).	
Передумови	E-mail вже зареєстровано в системі Користувач вийшов з акаунту
Кроки	1. Перейти на сторінку входу

Тест кейс 2. Вхід для існуючого користувача (правильні дані).	
	2. Ввести значення в поля форми 3. Натиснути на кнопку “Увійти”
Дані на вхід	Email = ‘alexandr.kirpichyov@gmail.com’ Пароль = ‘Qwerty7@’
Очікуваний результат	• Перехід на головну сторінку профілю користувача

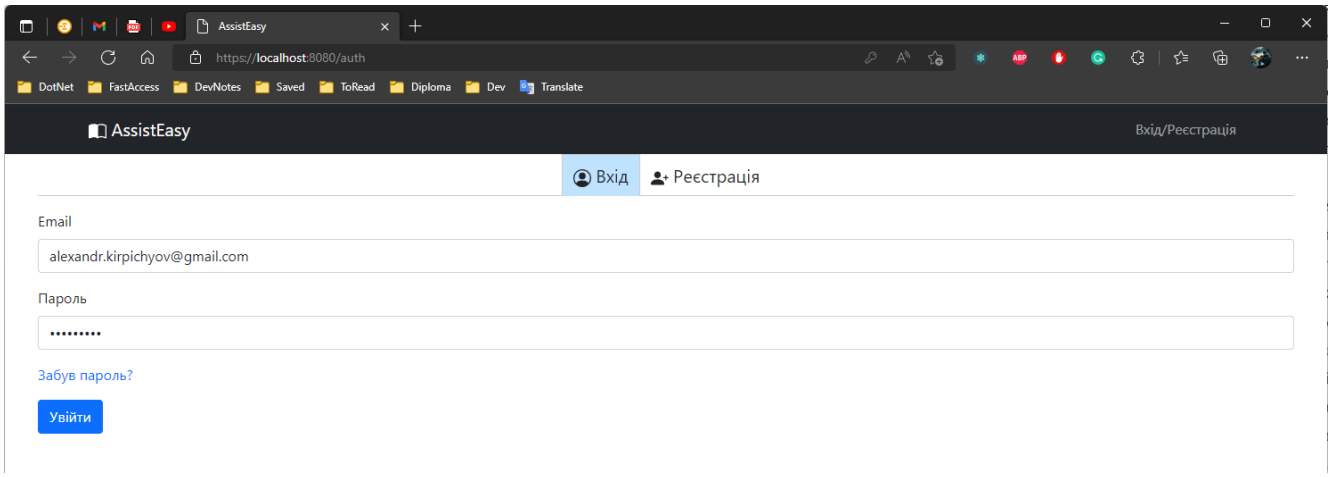


Рисунок 55. – Сторінка входу AssistEasy [Тест кейс 2]

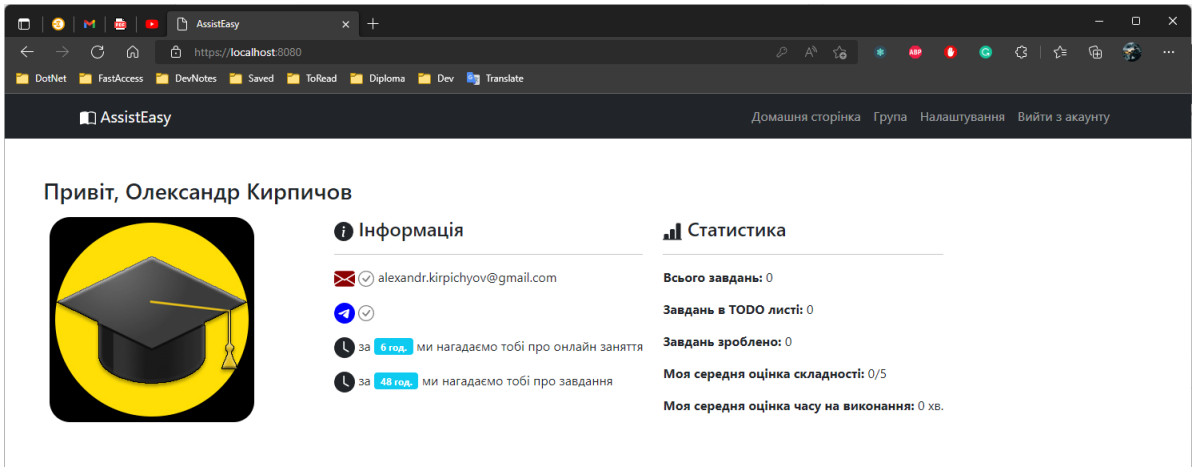


Рисунок 56. – Сторінка профілю AssistEasy [Тест кейс 2]

Таблиця 9 – Тест кейс 3. Вхід для існуючого користувача (неправильні дані)

Тест кейс 3. Вхід для існуючого користувача (неправильні дані).	
Передумови	E-mail вже зареєстровано в системі Користувач вийшов з акаунту
Кроки	1. Перейти на сторінку входу 2. Ввести значення в поля форми 3. Натиснути на кнопку “Увійти”
Дані на вхід	Email = ‘alexandr.kirpichyov@gmail.com’ Пароль = ‘1234567’
Очікуваний результат	• З’явилася помилка ‘Невірний email або пароль’ • Перехід на іншу сторінку не відбувся

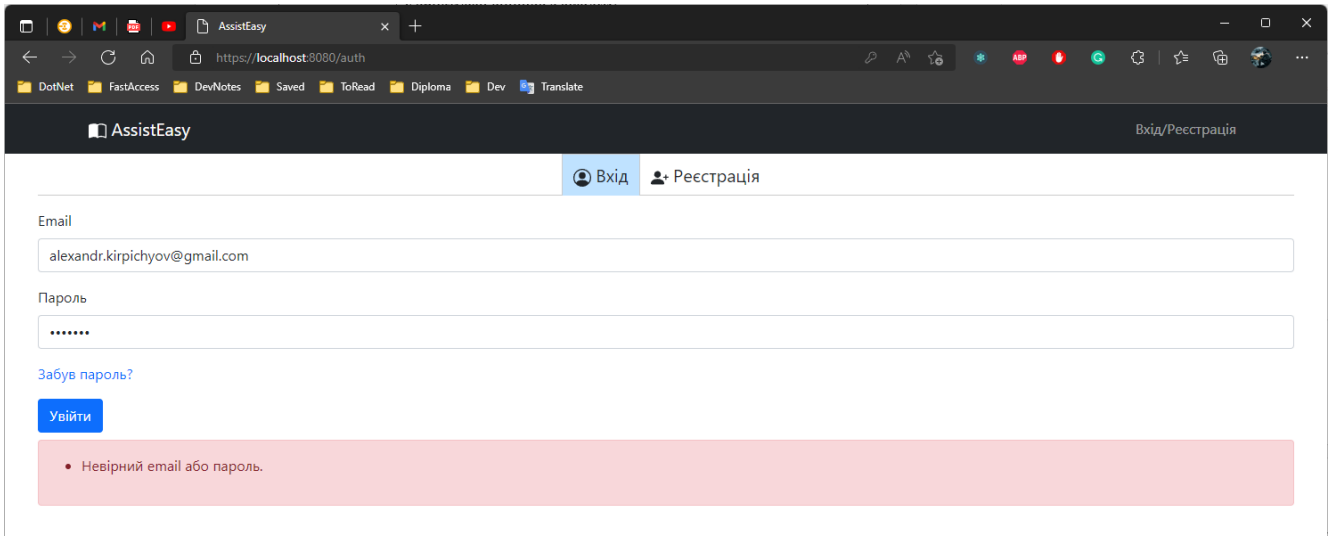


Рисунок 57. – Сторінка входу AssistEasy (валідація) [Тест кейс 3]

Таблиця 10 – Тест кейс 4. Створення групи

Тест кейс 4. Створення групи	
Передумови	Користувач увійшов в акаунт Користувач не має групи Користувач має підтверджений email
Кроки	1. Перейти на сторінку групи 2. Ввести значення в поля форми 3. Натиснути на кнопку “Створити”
Дані на вхід	Назва групи = ‘ПД-43’ Аватар групи = 
Очікуваний результат	• Сторінка оновилася та з’явилася інформація про створену групу • Група з’явилася в системі та БД

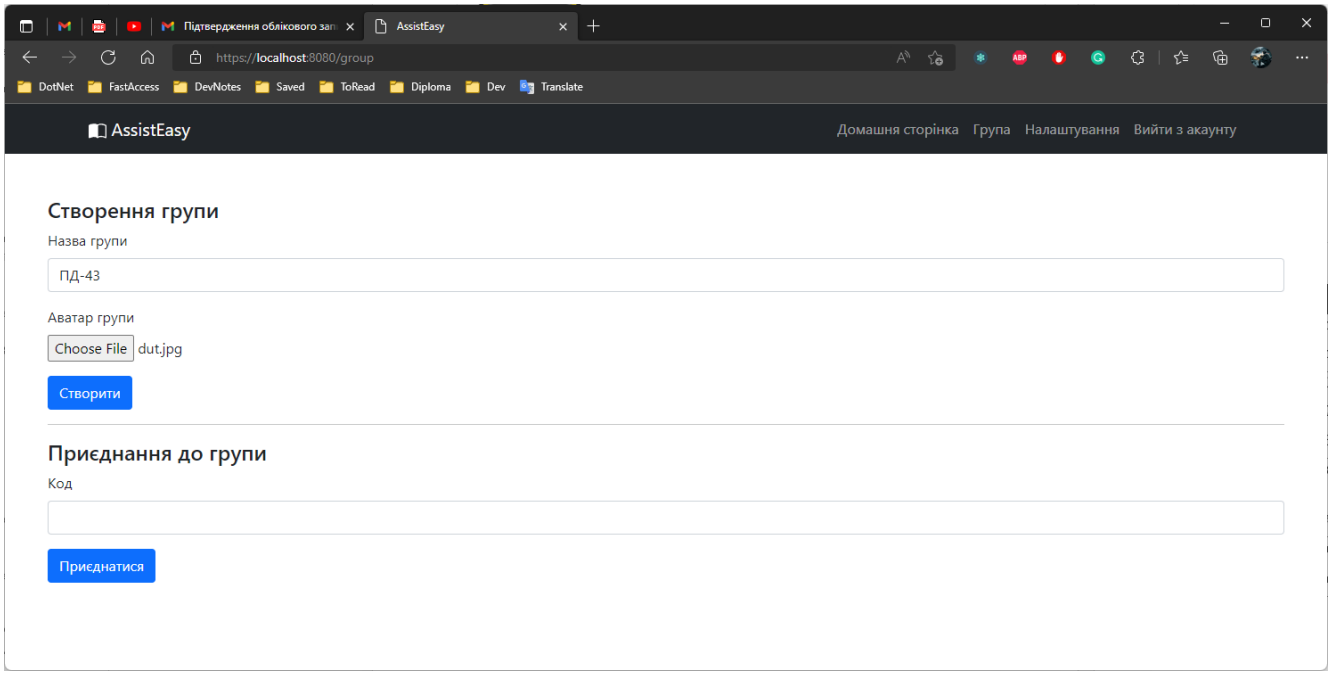


Рисунок 58. – Створення групи в AssistEasy [Тест кейс 4]

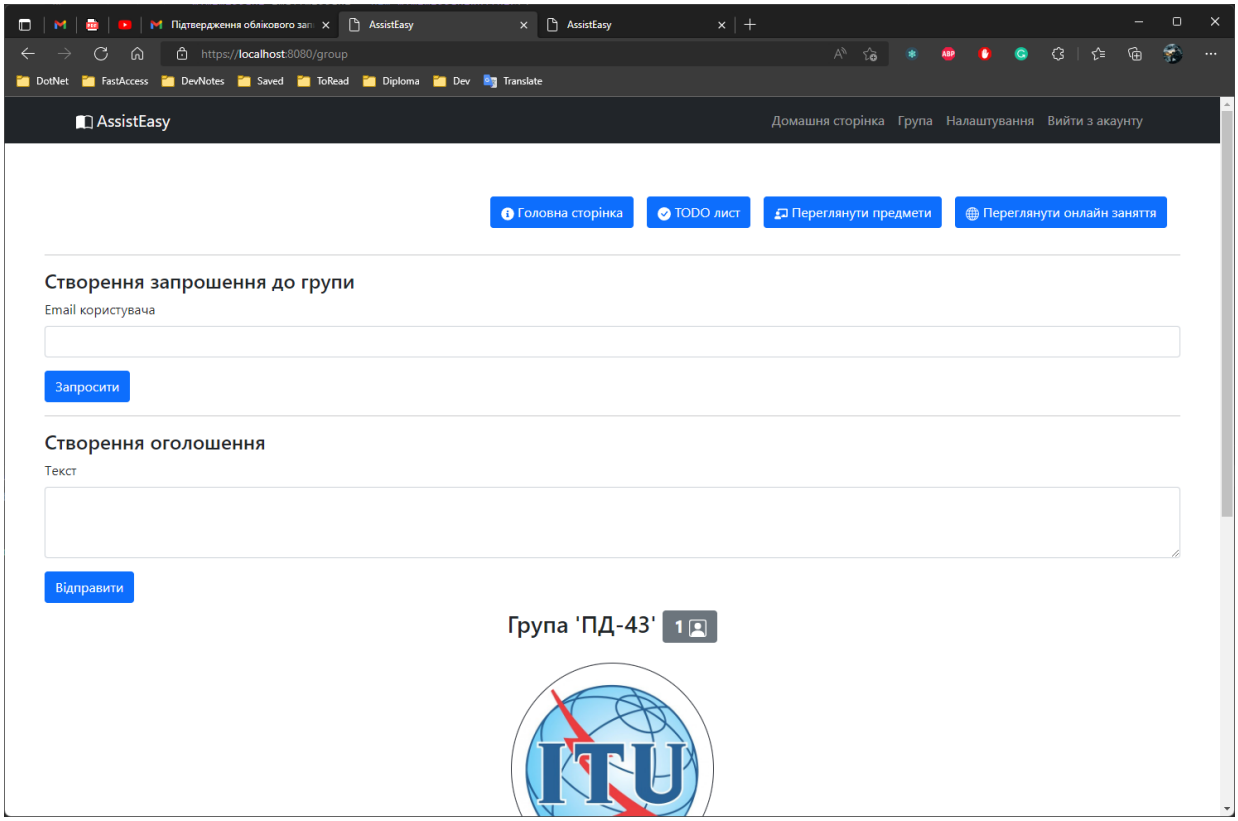


Рисунок 59. – Сторінка групи AssistEasy (адміністратор групи) [Тест кейс 4]

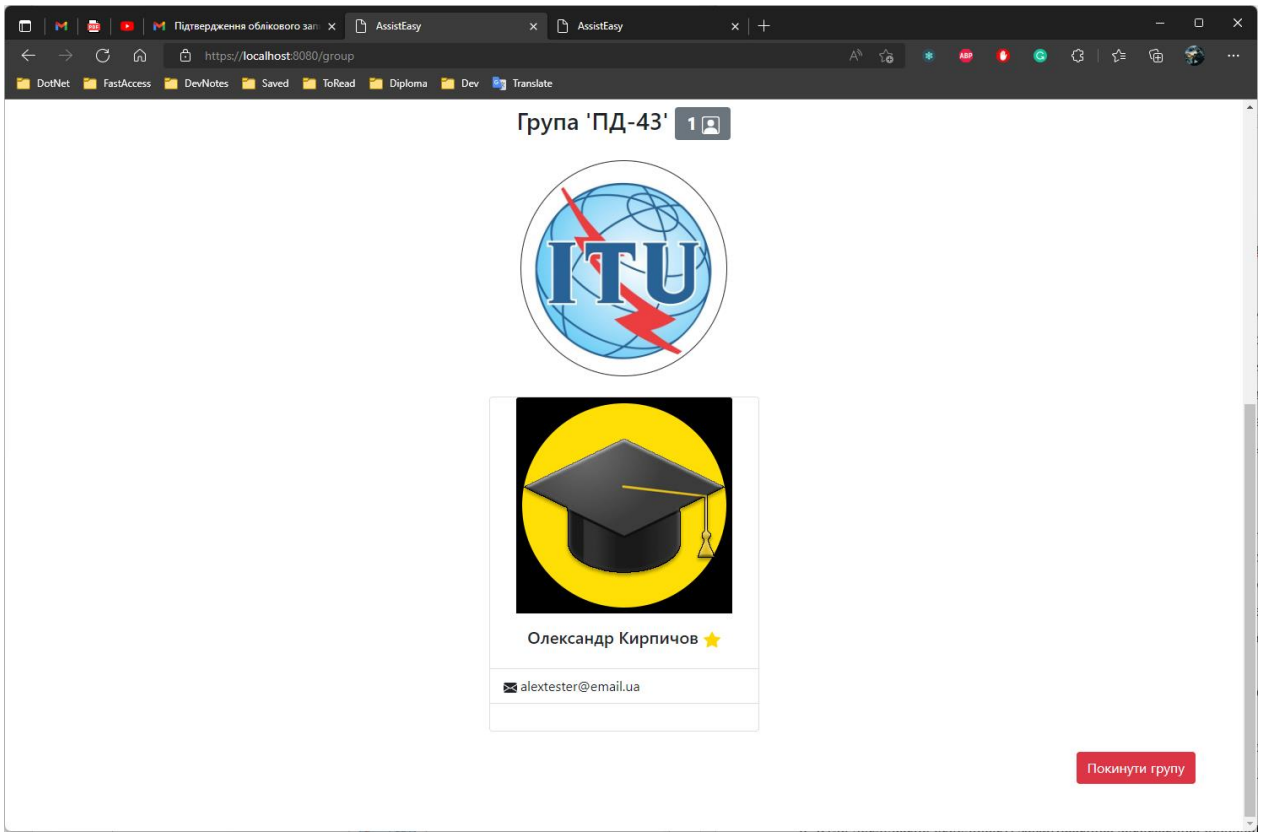


Рисунок 60. – Сторінка групи AssistEasy [Тест кейс 4]

Таблиця 11 – Тест кейс 5. Приєднання до групи

Тест кейс 5. Приєднання до групи	
Передумови	Групу створено Користувачі увійшли в акаунти Користувач, що приєднується, не має групи Користувачі мають підтверджений email Користувач 2 = alexandr.kirpichyov@gmail.com Користувач 1 = darkwhiteryb@gmail.com
Кроки	1. Перейти на сторінку групи (Користувач 1) 2. Ввести значення в поля форми “Створення запрошення до групи” (Користувач 1) 3. Натиснути на кнопку “Запросити” (Користувач 1) 4. Перейти на сторінку групи (Користувач 2) 5. Ввести код запрошення з email листа (Користувач 2) 6. Натиснути на кнопку “Приєднатися” (Користувач 2)
Дані на вхід	Email користувача = ‘darkwhiteryb@gmail.com’
Очікуваний результат	• Сторінка оновилася та з’явилася інформація про групу (Користувач 2) • Користувач 2 відобразився в списку членів групи • Дані членів групи в БД оновилися

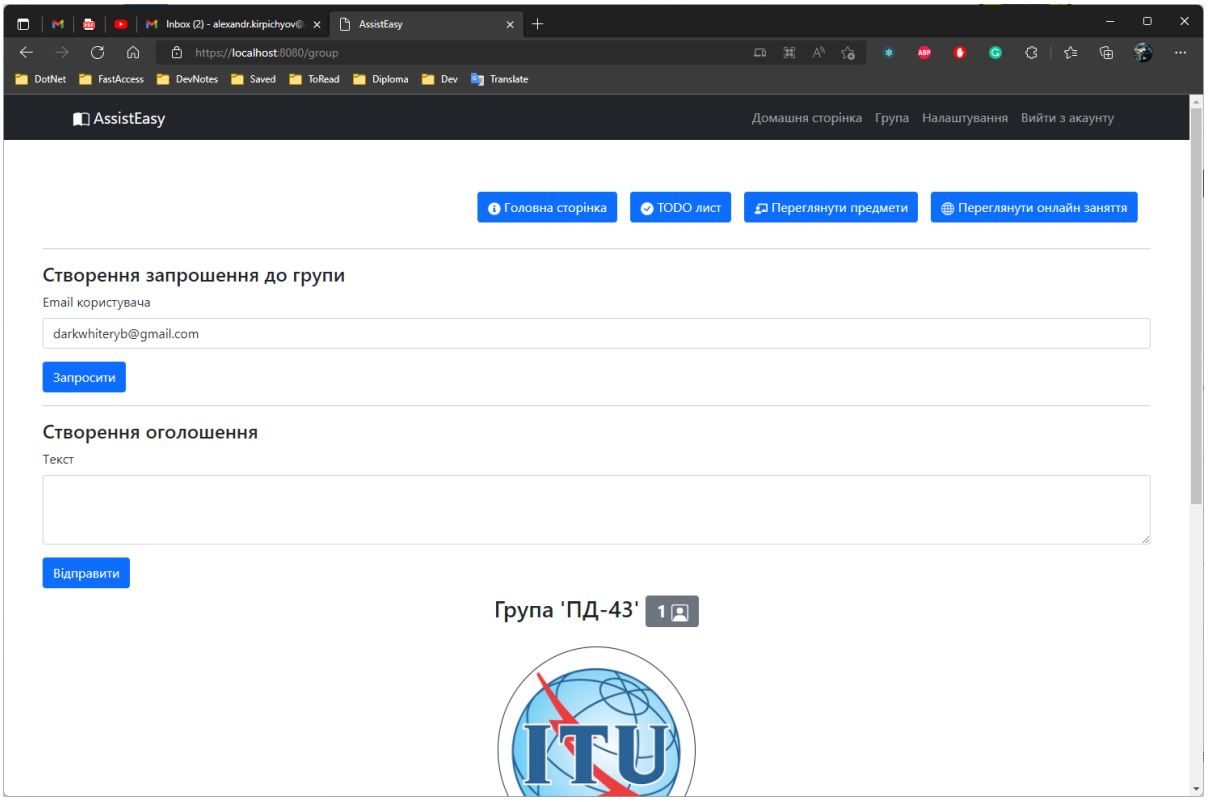


Рисунок 61. – Сторінка групи AssistEasy (адміністратор групи) [Тест кейс 5]

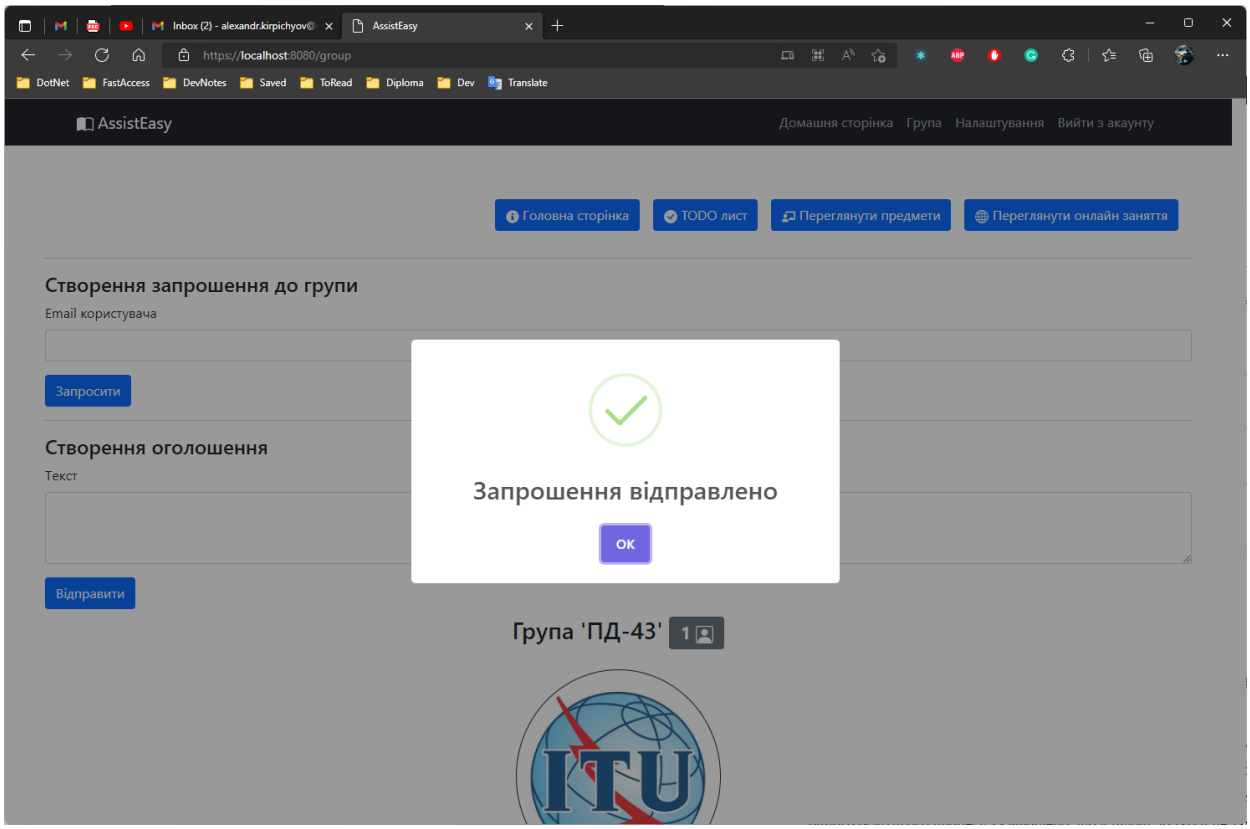


Рисунок 62. – Відправлення запрошення в групу AssistEasy [Тест кейс 5]

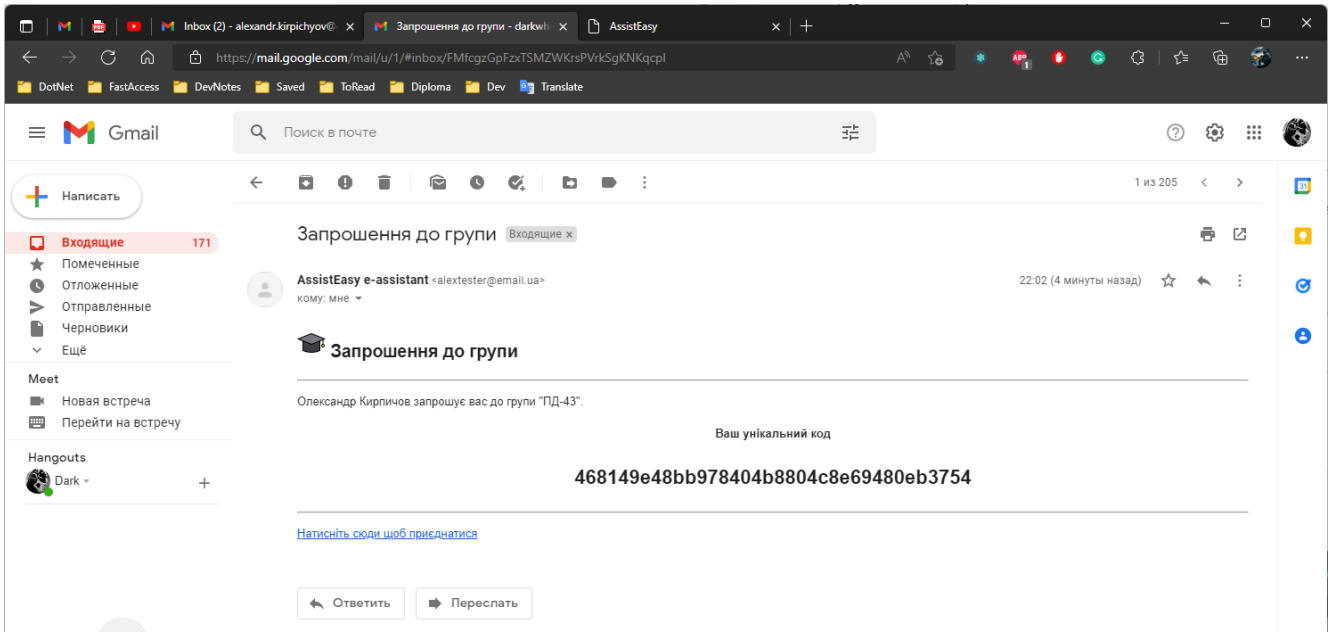


Рисунок 63. – Лист із запрошенням в групу AssistEasy [Тест кейс 5]

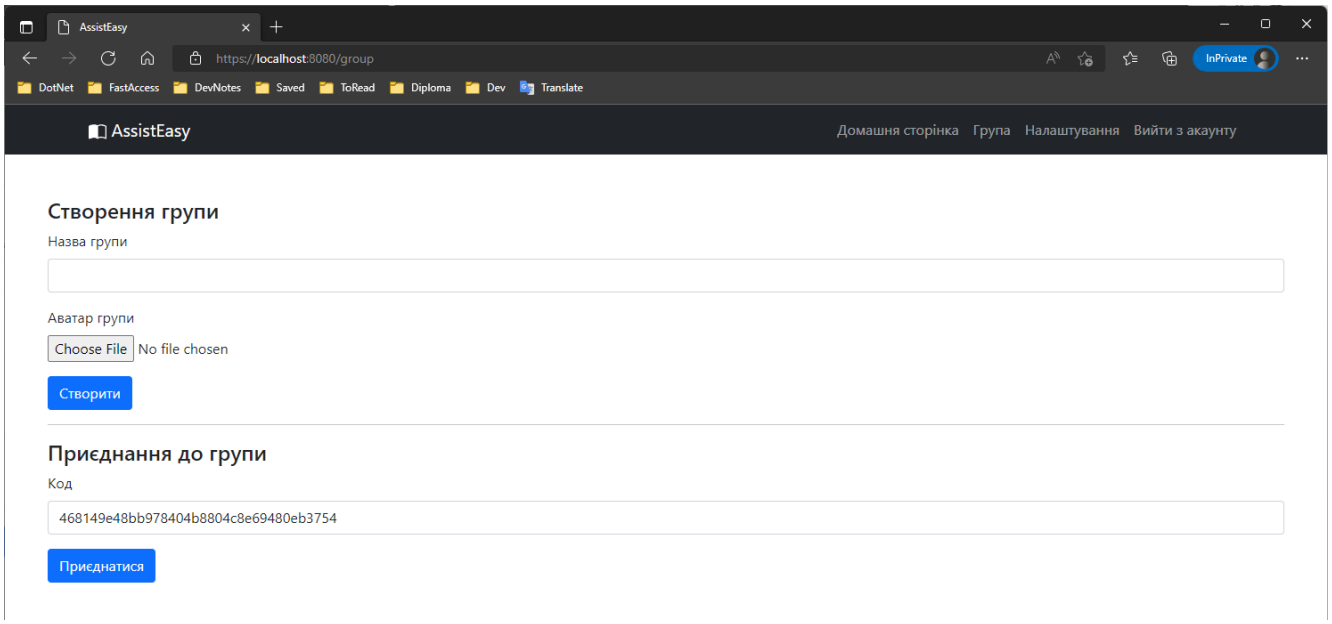


Рисунок 64. – Сторінка приєднання до групи AssistEasy [Тест кейс 5]

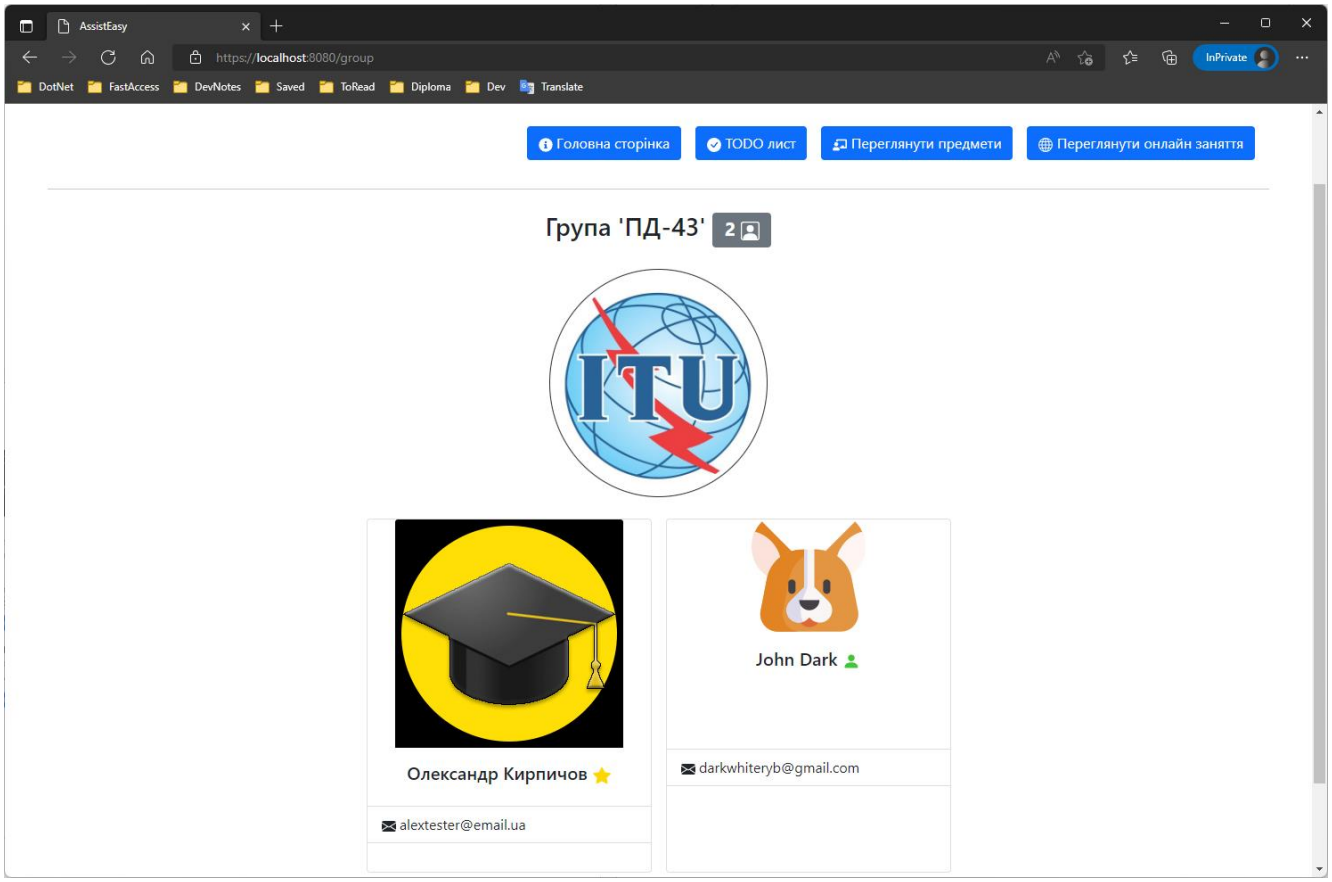


Рисунок 65. – Сторінка групи AssistEasy [Тест кейс 5]

Таблиця 12 – Тест кейс 6. Прив'язання телеграм акаунту

Тест кейс 6. Прив'язання телеграм акаунту	
Передумови	Користувач увійшов в акаунт Користувач має підтверджений email
Кроки	1. Відкрити чат з telegram ботом 2. Натиснути start (або ввести команду /start) 3. Натиснути на /code (або ввести команду вручну) 4. Перейти на сторінку налаштувань 5. Натиснути на кнопку "Прив'язати telegram" 6. Ввести отриманий від бота код в поле "Унікальний код" та натиснути кнопу "Прив'язати"
Дані на вхід	Код, отриманий від бота

Тест кейс 6. Прив’язання телеграм акаунту	
Очікуваний результат	• Отримано повідомлення від бота про успішне прив’язання • Інформація про користувача оновилася в системі та БД

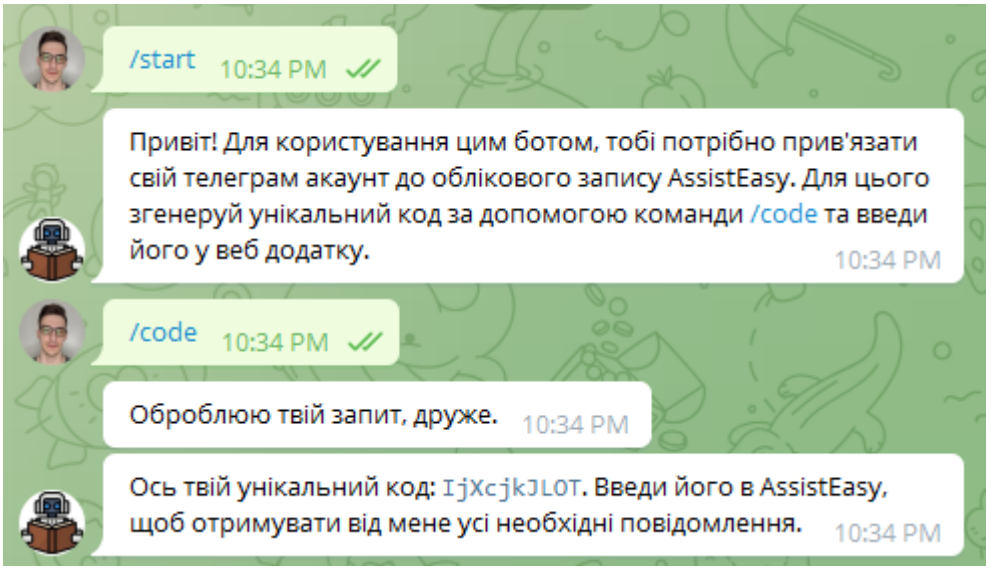


Рисунок 66. – Отримання коду в telegram бота AssistEasy [Тест кейс 6]

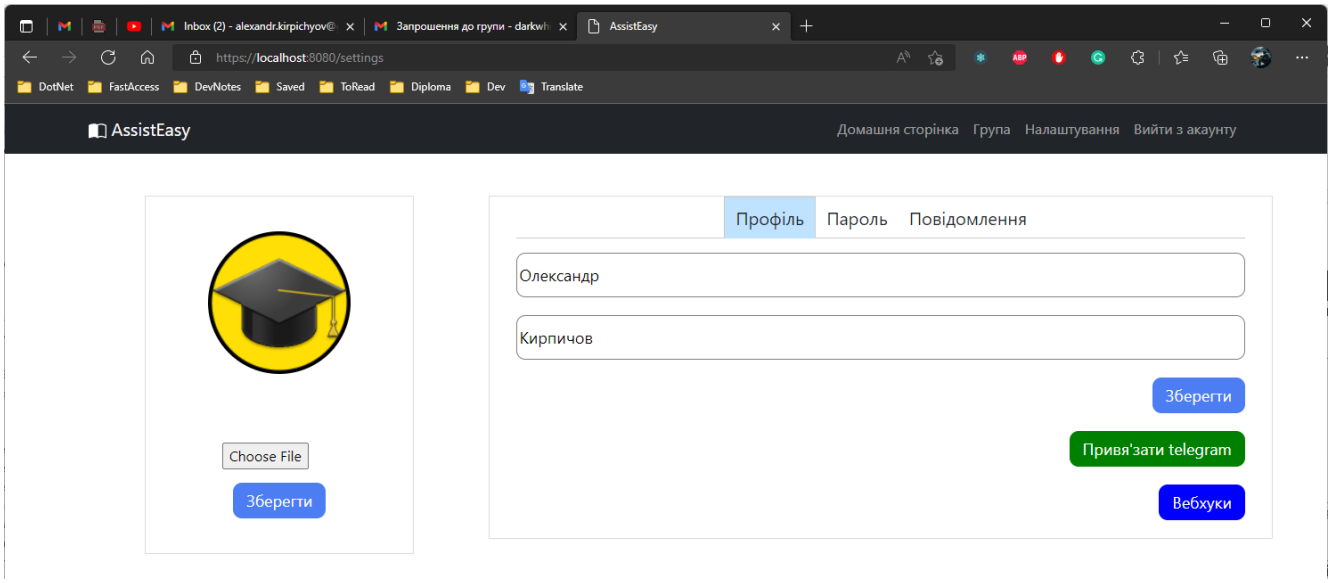


Рисунок 67. – Сторінка налаштувань профілю AssistEasy [Тест кейс 6]

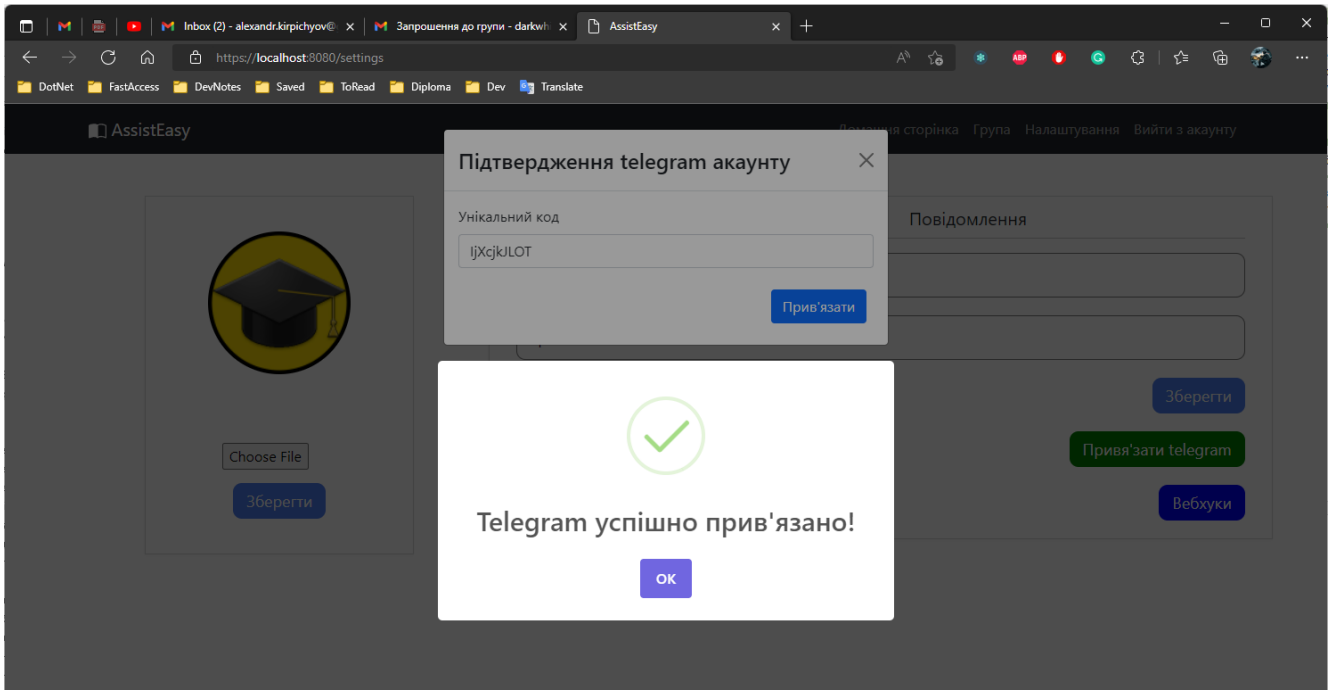


Рисунок 68. – Прив'язання telegram до AssistEasy [Тест кейс 6]

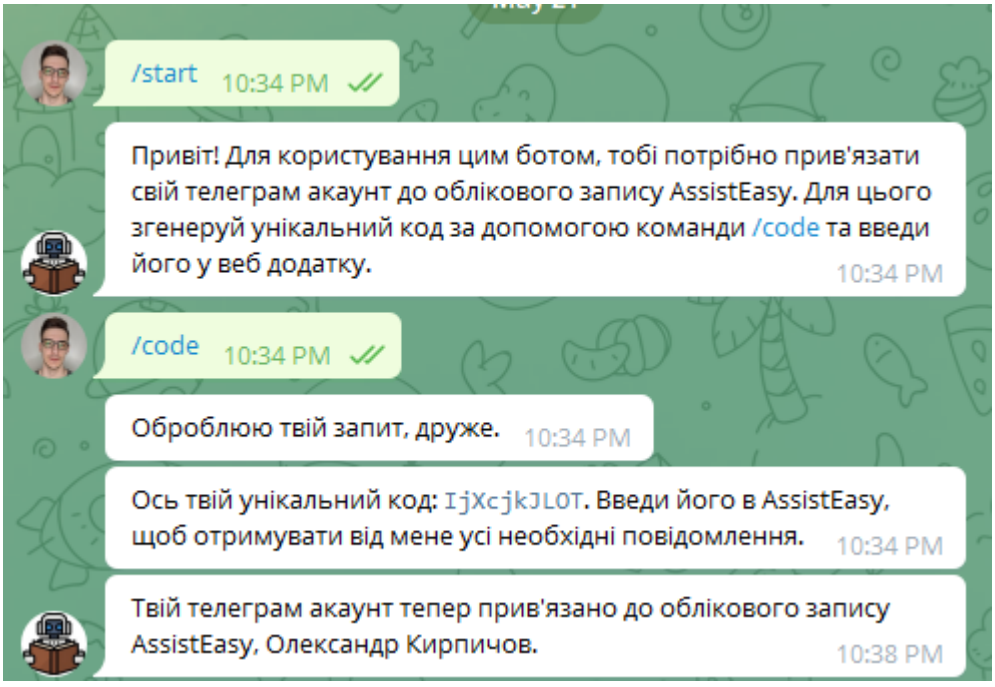


Рисунок 69. – Повідомлення про успішне підтвердження від бота AssistEasy [Тест кейс 6]

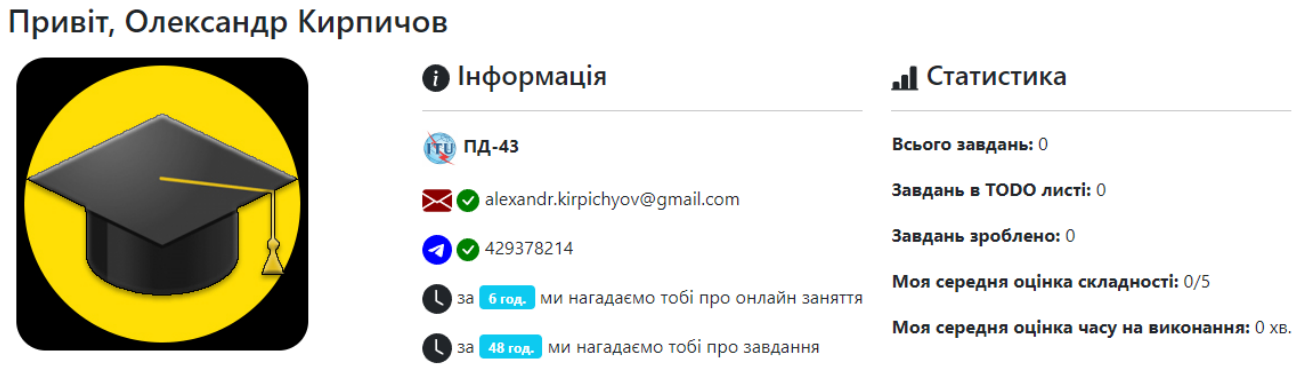


Рисунок 70. – Сторінка профілю AssistEasy [Тест кейс 6]

Таблиця 13 – Тест кейс 7. Створення предмету та завдання

Тест кейс 7. Створення предмету та завдання	
Передумови	Користувач увійшов в акаунт Користувач є власником або адміністратором групи Користувач підтвердив телеграм акаунт та має увімкнуті повідомлення
Кроки	1. Перейти на сторінку групи 2. Натиснути на кнопку “Переглянути предмети” 3. Натиснути на кнопку “Створити” 4. Заповнити поля форми створення предмету 5. Натиснути на кнопку “Зберегти” 6. Натиснути на кнопку завдань з предмету 7. Натиснути на кнопку “Додати завдання” 8. Заповнити поля форми додання завдання 9. Натиснути на кнопку “Зберегти”
Дані на вхід	**Форма створення предмету** Назва = ‘ООП C#’ Ім'я та прізвище викладача = ‘Гаманюк І.М.’ Опис = ‘Дуже цікавий предмет!’

Тест кейс 7. Створення предмету та завдання	
	Форма створення завдання Назва = ‘Створити консольну бібліотеку’ Опис = ‘Потрібно створити бібліотеку для кольорового виводу тексту в консоль’ Тип = лабораторна Дата = наступний день
Очікуваний результат	• Предмет та завдання з’явилася в системі та БД • Отримано оголошення в телеграм боті та email

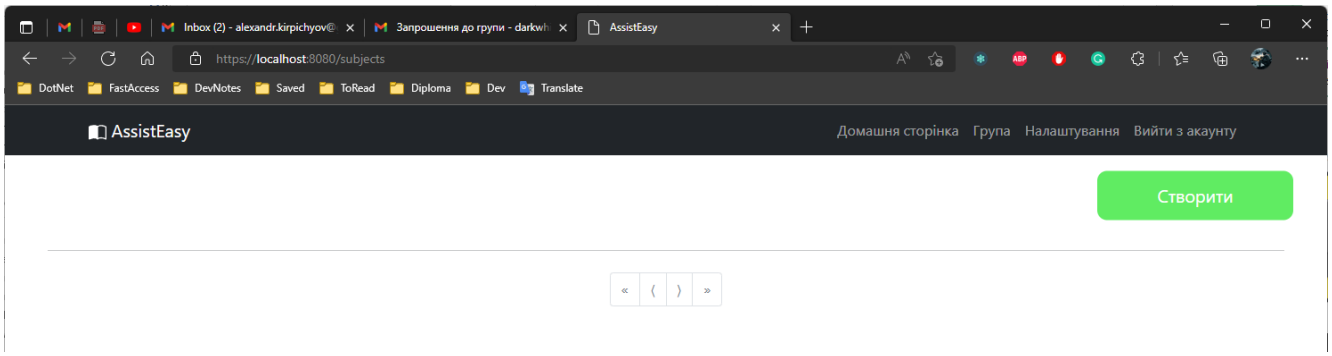


Рисунок 71. – Сторінка предметів AssistEasy [Тест кейс 7]

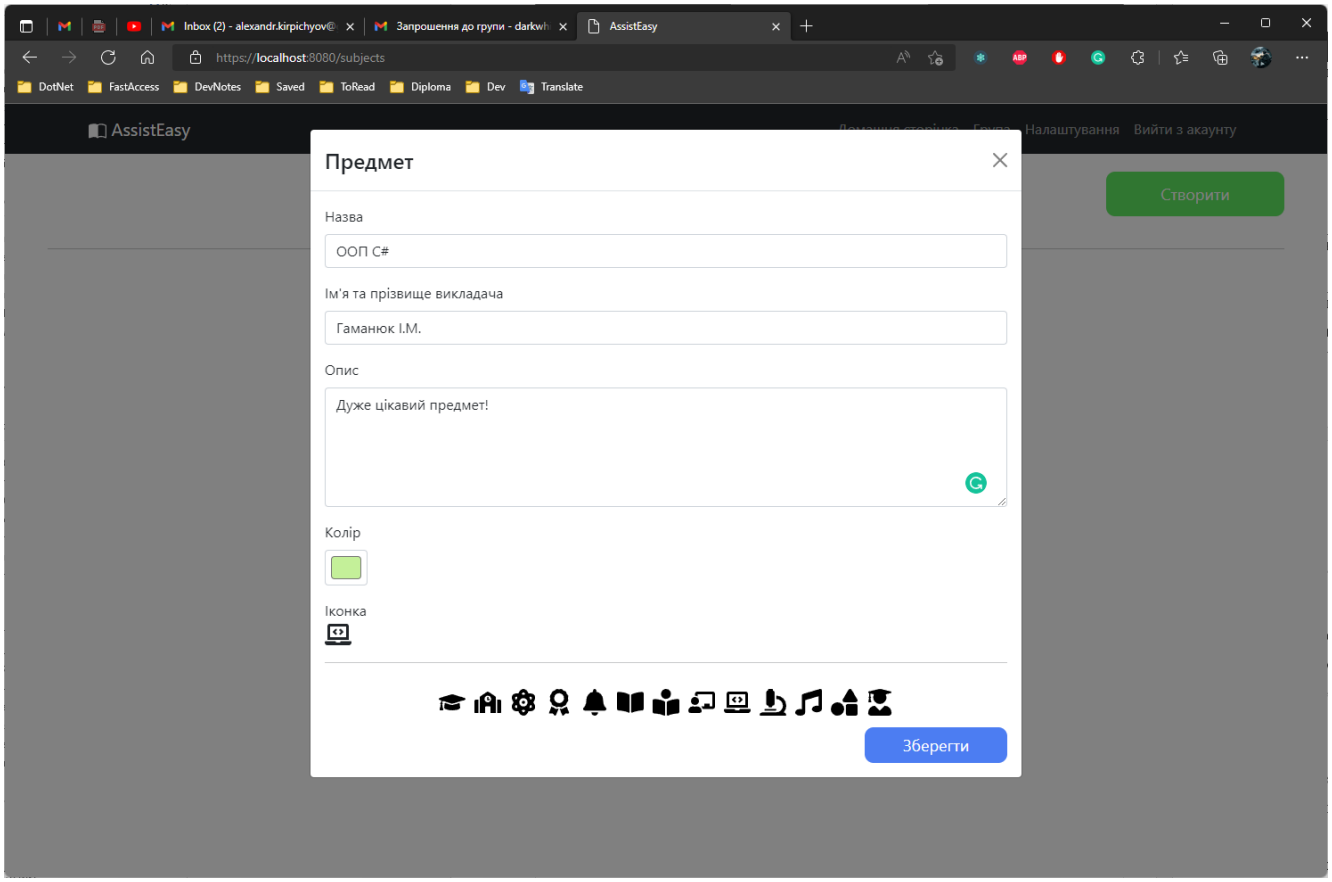


Рисунок 72. – Форма створення предмету AssistEasy [Тест кейс 7]

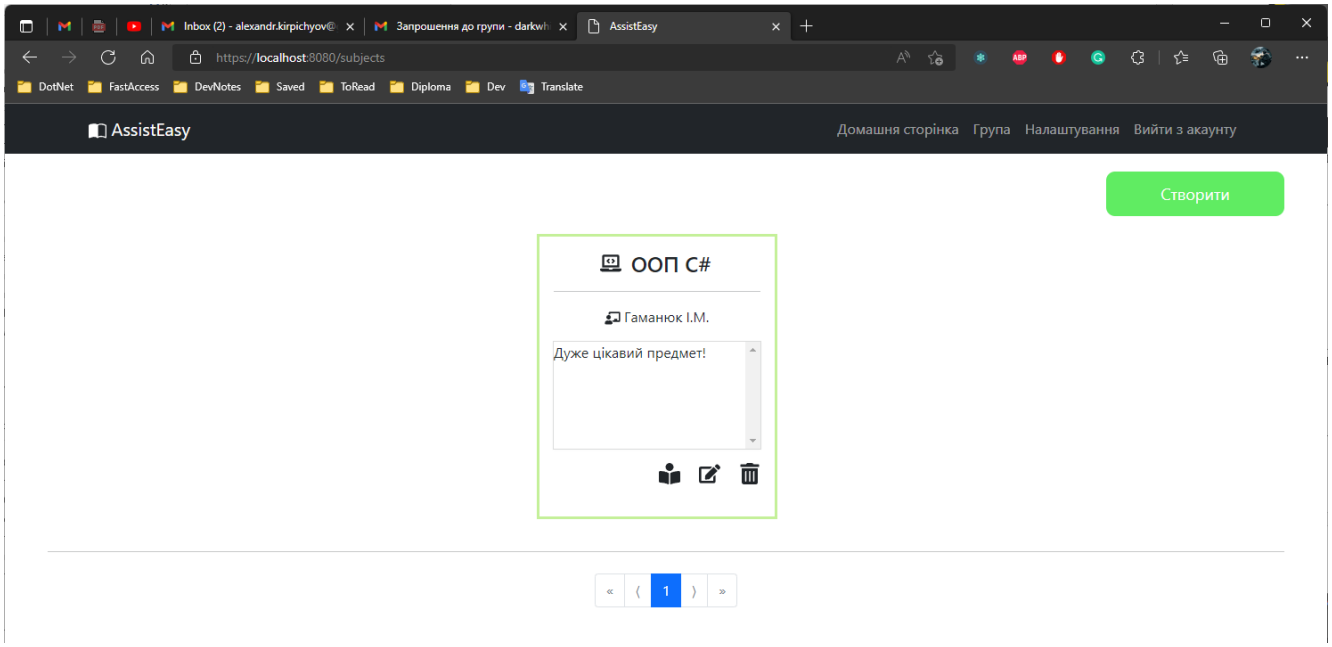


Рисунок 73. – Сторінка предметів AssistEasy [Тест кейс 7]

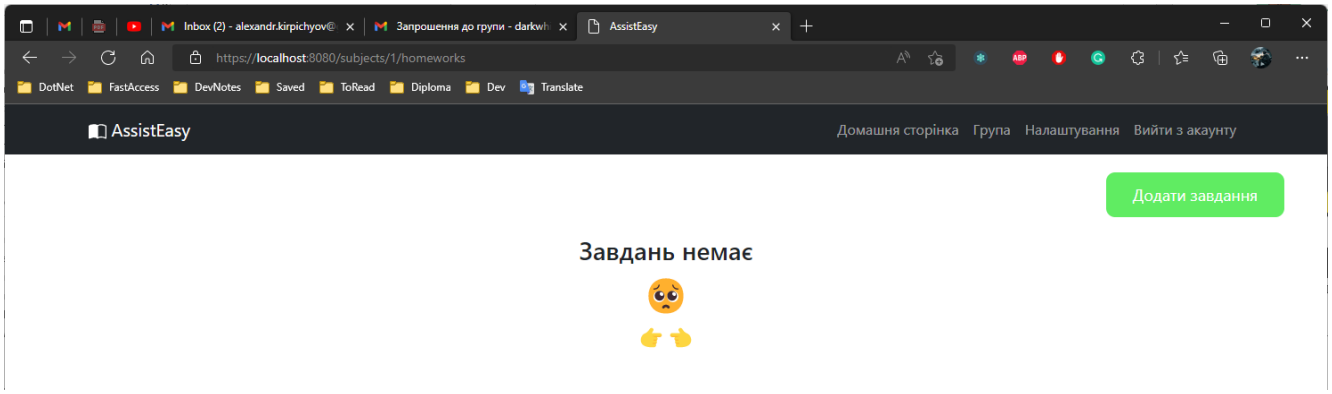


Рисунок 74. – Сторінка завдань AssistEasy [Тест кейс 7]

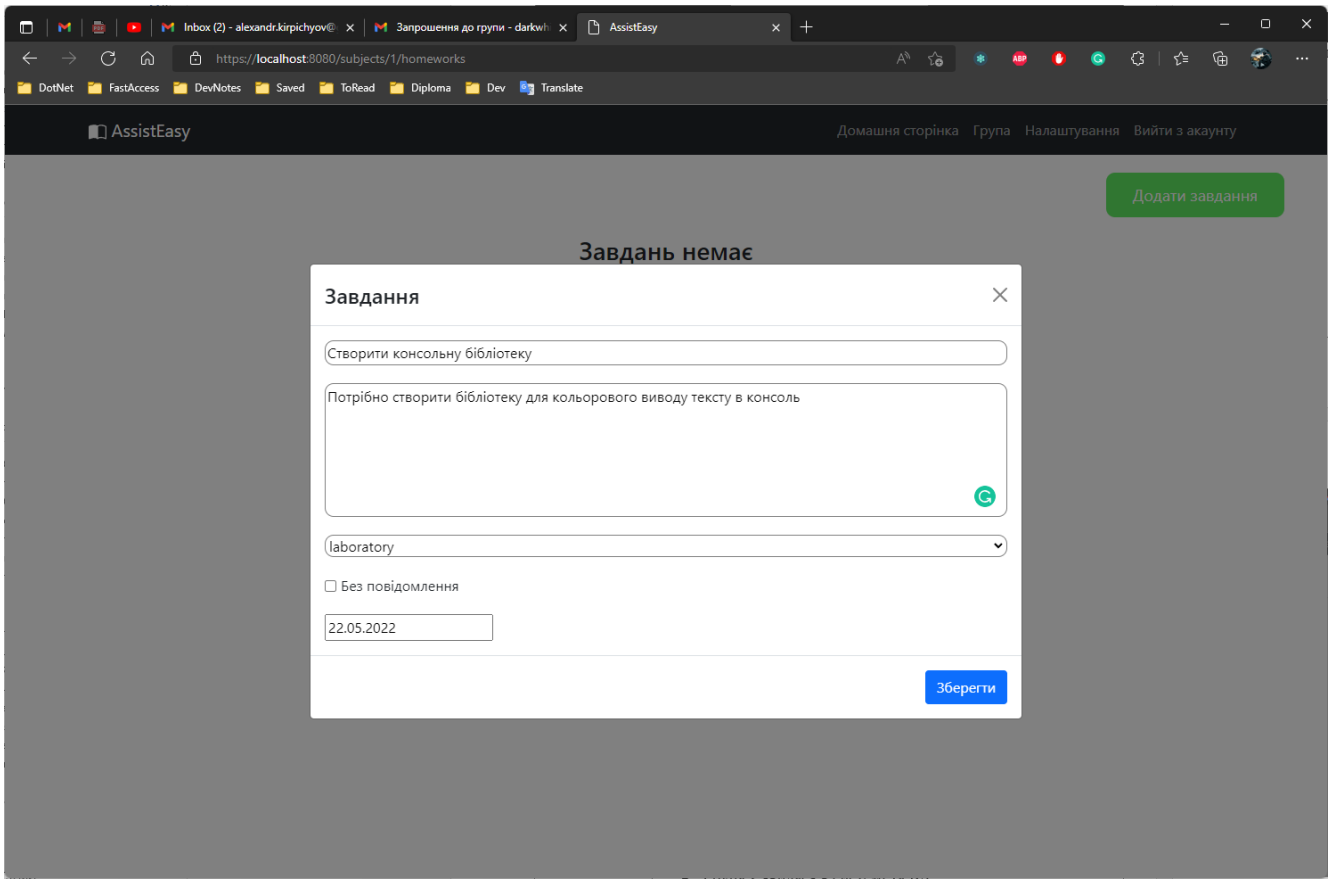


Рисунок 75. – Форма створення завдання AssistEasy [Тест кейс 7]

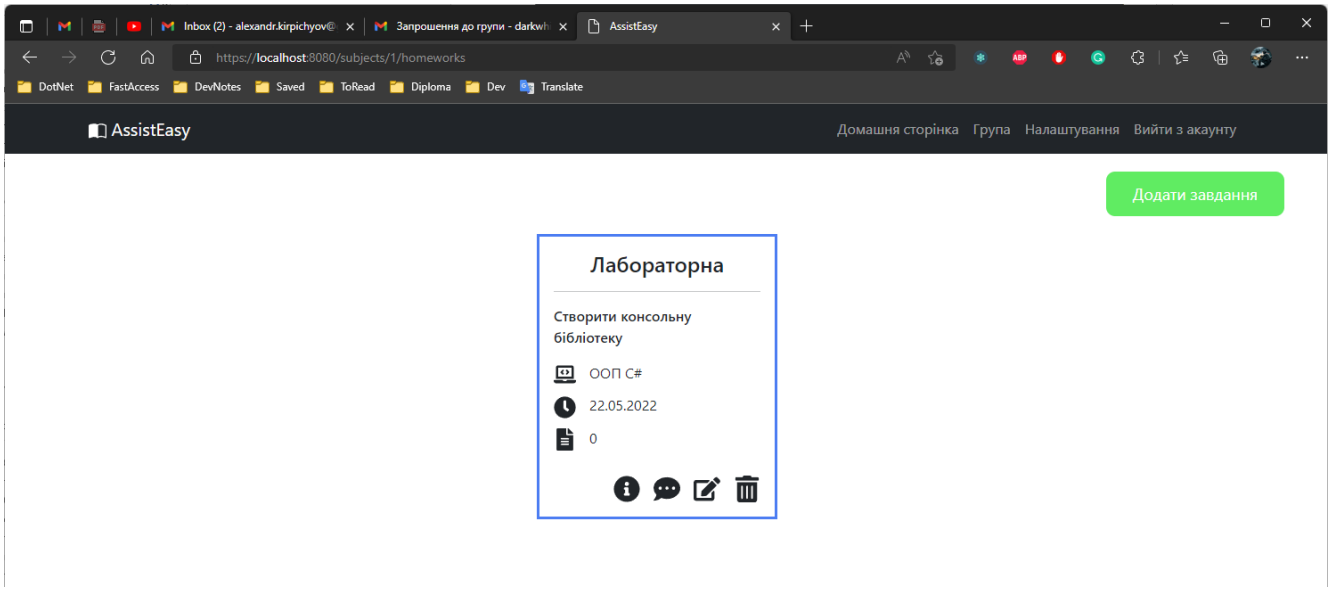


Рисунок 76. – Сторінка завдань AssistEasy [Тест кейс 7]

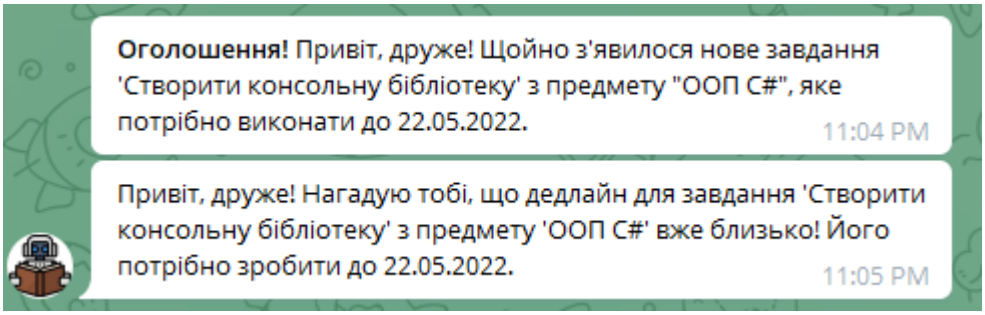


Рисунок 77. – Повідомлення від telegram бота AssistEasy [Тест кейс 7]

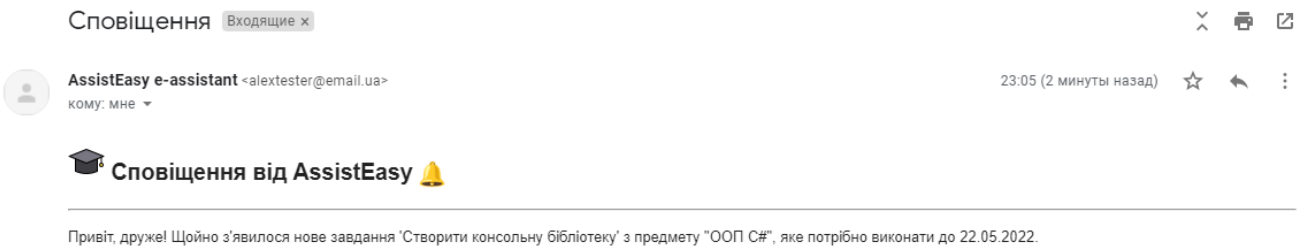


Рисунок 78. – Лист з повідомленням від AssistEasy [Тест кейс 7]

Таблиця 14 – Тест кейс 8. Додання файлів до завдання

Тест кейс 8. Додання файлів до завдання	
Передумови	Користувач увійшов в акаунт Користувач є власником або адміністратором групи Завдання вже створене
Кроки	1. Перейти на сторінку завдань предмету 2. Натиснути на кнопку редагування предмету 3. Натиснути на кнопку “Choose file” 4. Обрати файл 5. Натиснути на кнопку “Додати файл” 6. Натиснути на кнопку “Зберегти” 7. Натиснути на кнопку детальної інформації 8. Натиснути на доданий файл
Дані на вхід	Файл = ‘Дипломникам.pdf’
Очікуваний результат	• Файл завдання з’явився в системі та БД • Файл завантажується при кліку на нього та успішно відкривається

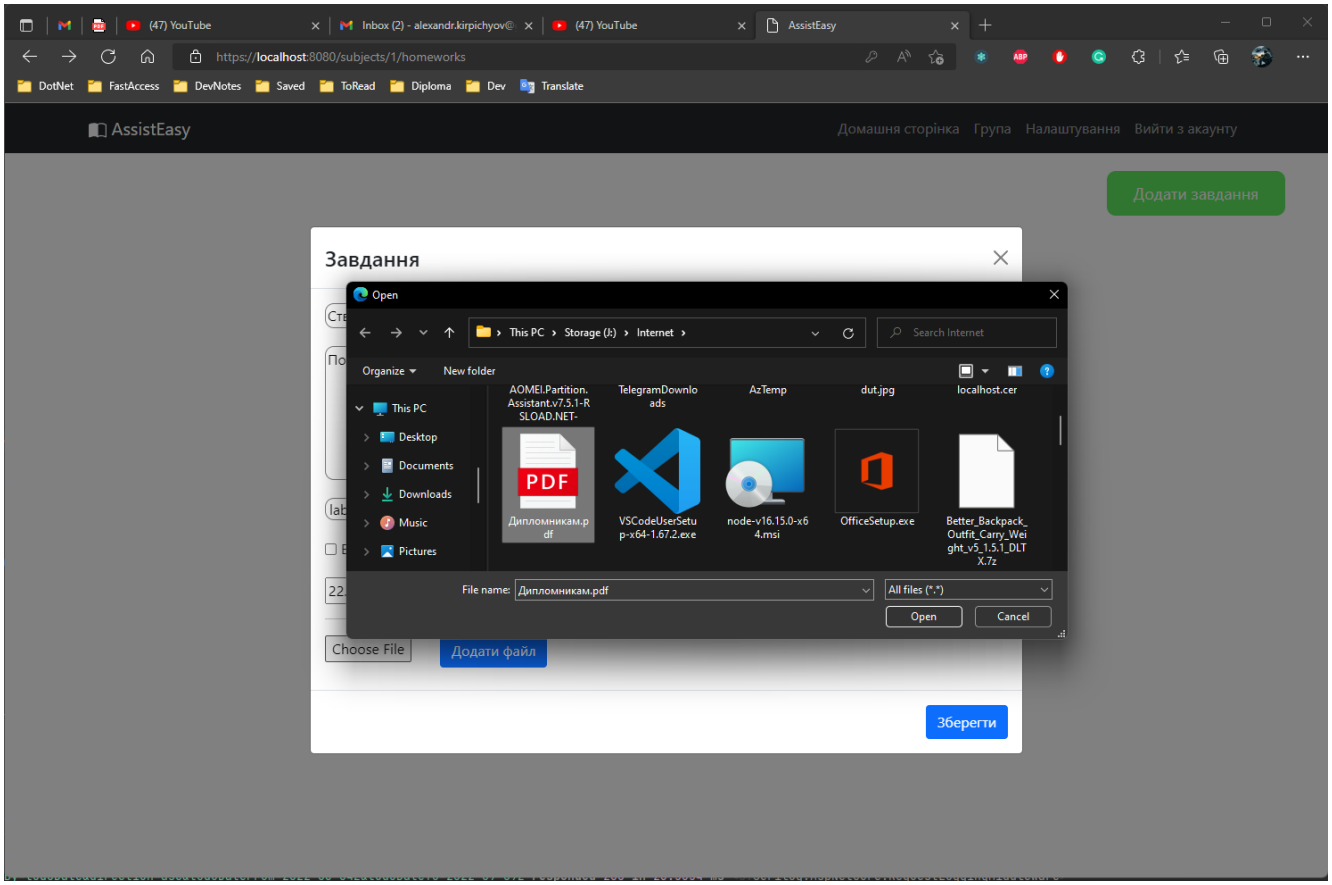


Рисунок 79. – Форма додання файлу до завдання AssistEasy [Тест кейс 8]

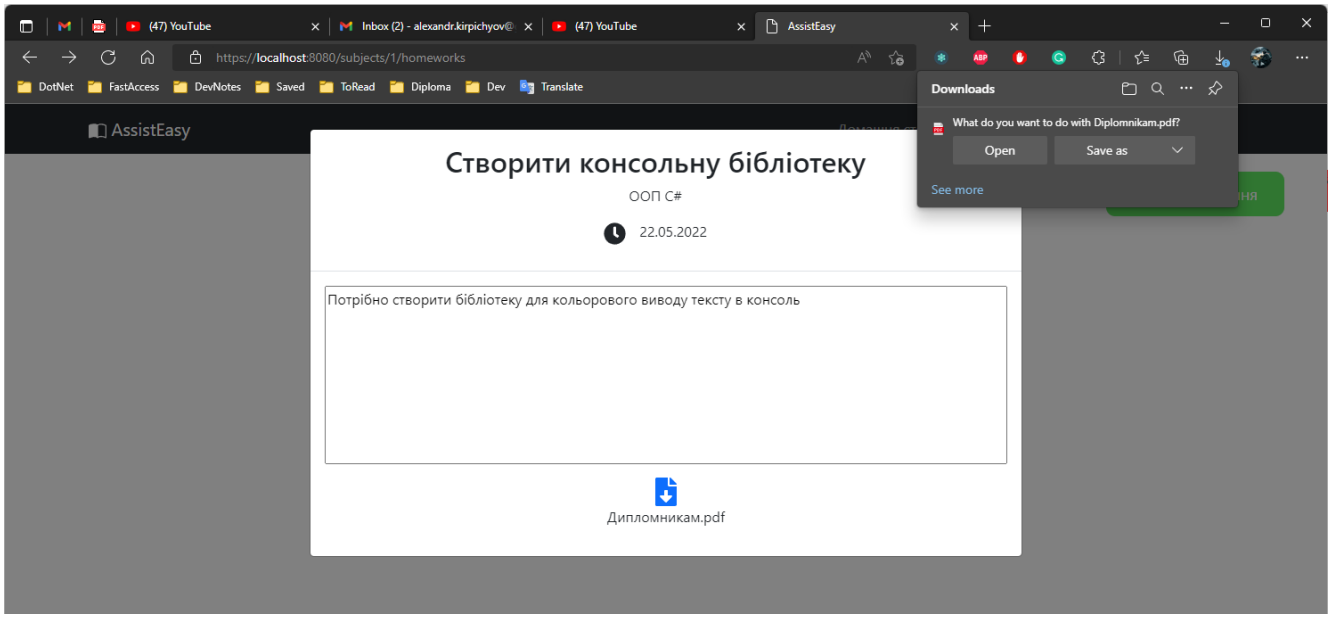


Рисунок 80. – Завантаження доданих файлів завдання AssistEasy [Тест кейс 8]

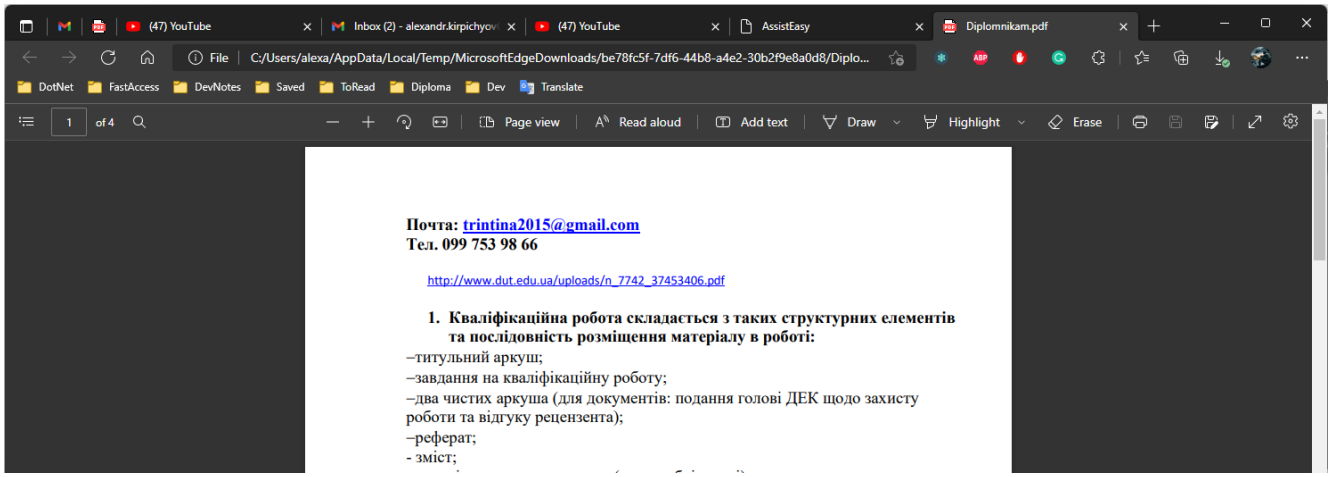


Рисунок 81. Завантажений файл з AssistEasy [Тест кейс 8]

Таблиця 15 – Тест кейс 9. Додання фідбеку до завдання

Тест кейс 9. Додання фідбеку до завдання	
Передумови	Користувач увійшов в акаунт Користувач має доступ до групи Завдання вже створене
Кроки	1. Перейти на сторінку завдань предмету 2. Натиснути на кнопку коментування предмету 💬 3. Ввести текст коментаря 4. Натиснути на кнопку відправлення 5. Натиснути кнопку “Оцінити” 6. Натиснути на відповідну кількість зірочок та обрати час виконання 7. Натиснути кнопку “Відправити”
Дані на вхід	Коментар = ‘Це було просто та цікаво! Тільки передивіться спочатку рекомендовані посилання.’ Складність = 2 зірочки Зробив за = 60 хвилин

Тест кейс 9. Додання фідбеку до завдання	
Очікуваний результат	• Коментар до завдання з’явився в системі та БД • Рейтинг складності та час виконання оновився

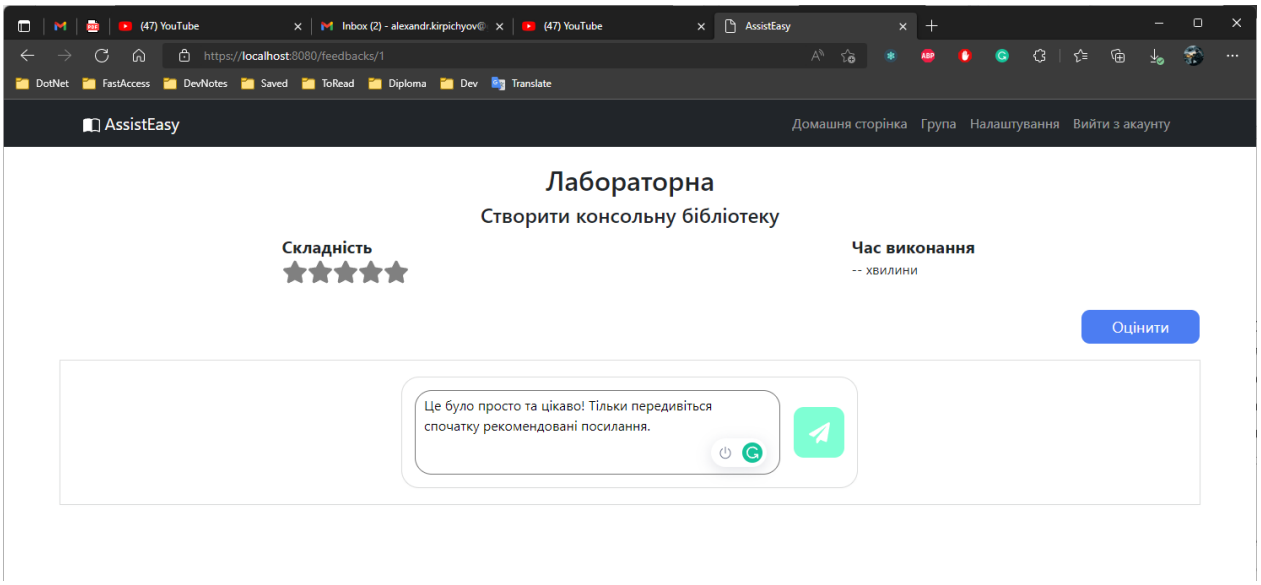


Рисунок 82. – Сторінка фідбеку до завдання AssistEasy [Тест кейс 9]

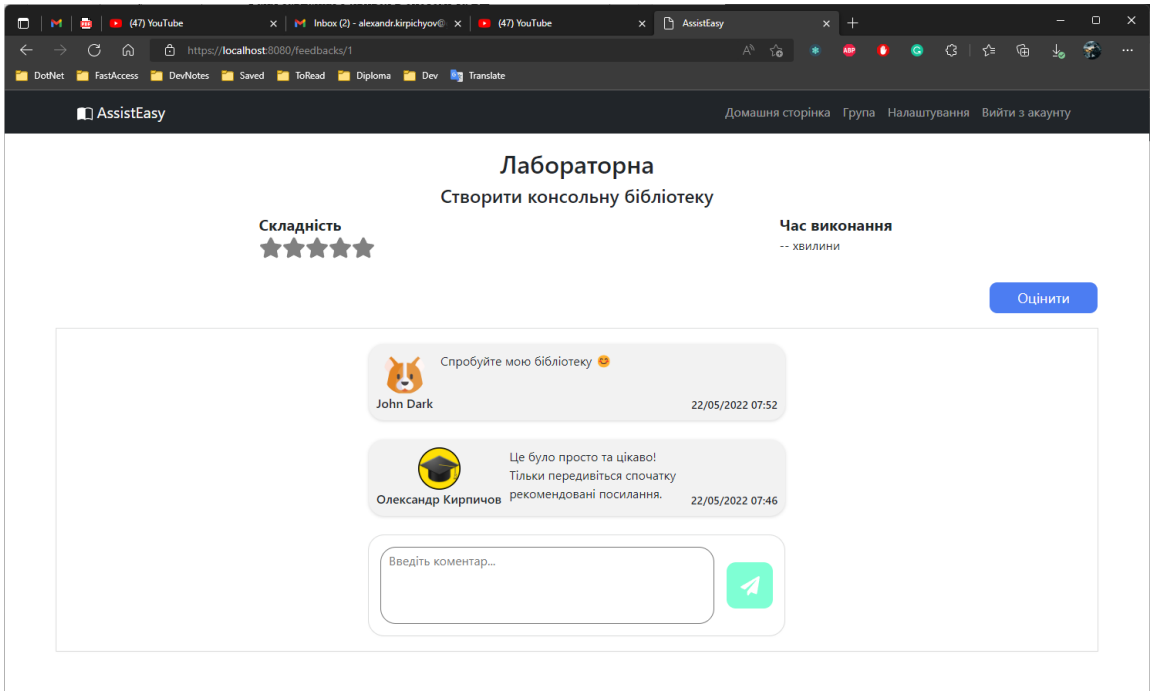


Рисунок 83. – Сторінка фідбеку до завдання AssistEasy (після додання коментаря) [Тест кейс 9]

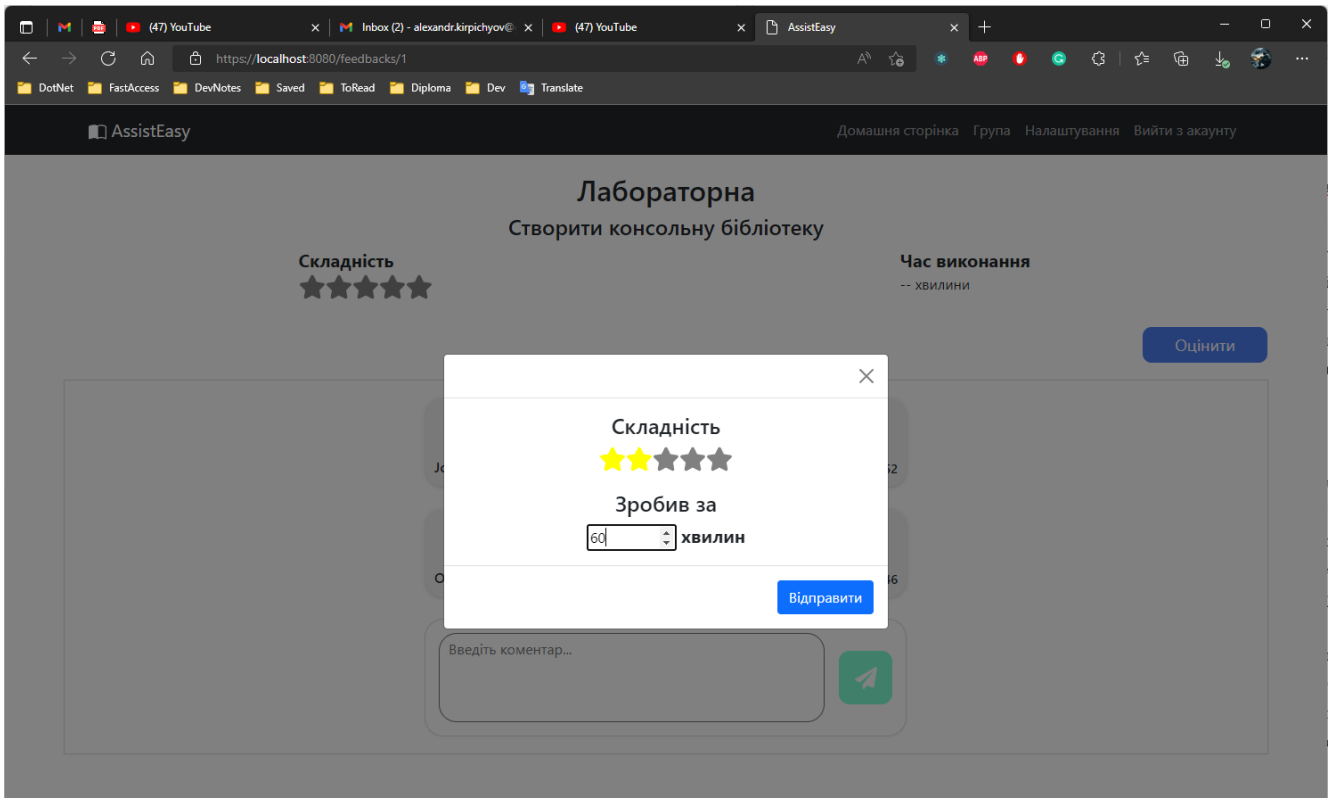


Рисунок 84. Форма фідбеку до завдання AssistEasy [Тест кейс 9]

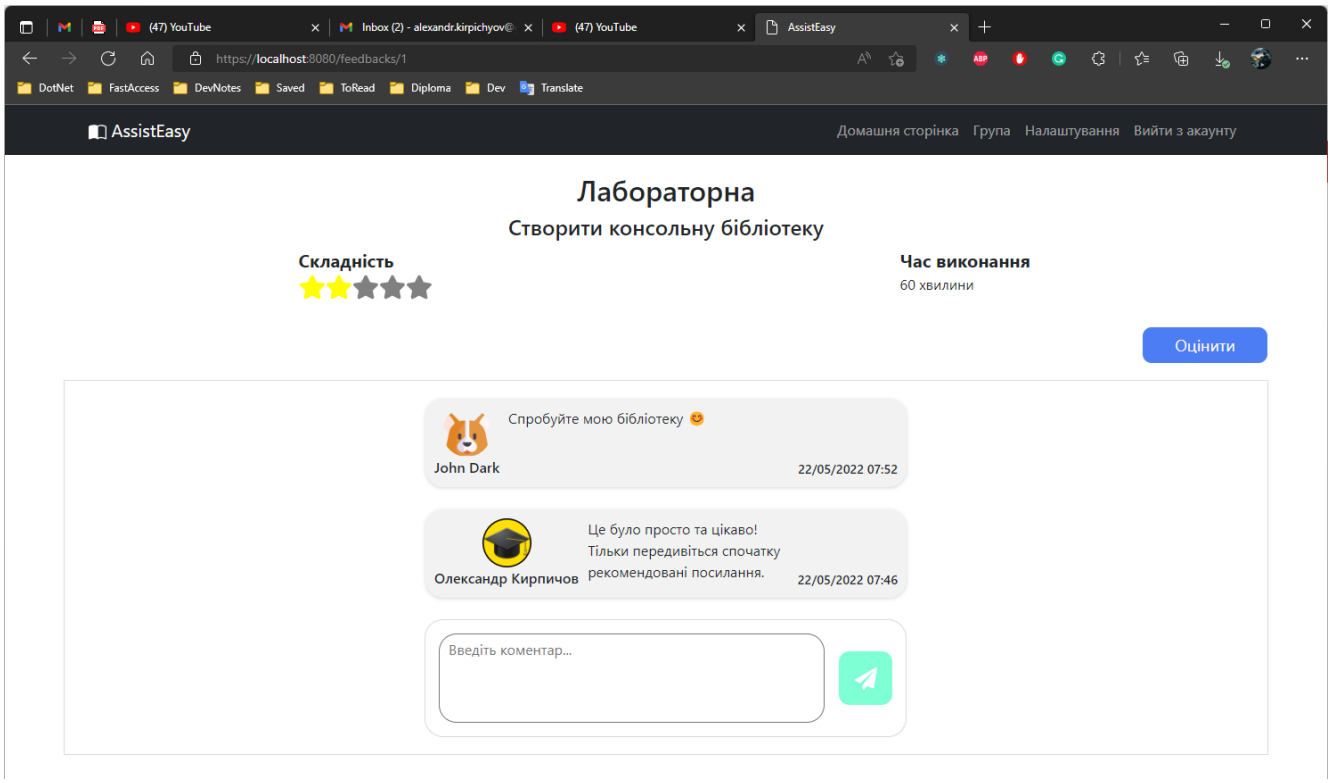


Рисунок 85. – Сторінка фідбеку до завдання AssistEasy (після оцінки складності) [Тест кейс 9]

Висновки

Дана робота була спрямована на проектування та реалізацію вебсервісу для студентів та учнів.

1. Було обґрунтовано актуальність розробленої системи та її наукову новизну шляхом аналізу близьких за функціоналом продуктів. Було досліджено типові проблеми, з якими зустрічаються студенти та учні під час дистанційного та очного освітнього процесу та виявлено, що існуючі додатки не задовольняють в повній мірі вимоги користувачів та спрямовані більше на викладачів ніж учнів. Було сформовано та описано функціональні та нефункціональні вимоги до системи.
2. Було спроектовано архітектуру системи та розроблено вебсервіс з використанням мікросервісів, сервісів Azure та брокеру повідомлень. В якості інструментів розробки було обрано мову програмування C# версії 9.0 в парі із фреймворком ASP.NET Core 5, бази даних PostgreSQL та MongoDB, сховище файлів Azure Blob storage, брокер повідомлень RabbitMQ, систему керування версіями Git, середовище розробки JetBrains Rider 2021.3, платформу Docker та наведено головні переваги цих інструментів.
3. Було описано архітектуру, ключові особливості та алгоритми роботи програмного забезпечення у виді схем та діаграм UML. Спроектована архітектура була представлена колегам та висококваліфікованим фахівцям з розробки програмного забезпечення компанії SoftServe рівню Middle та Senior, де отримала схвалення.
4. Було досліджено особливості завантаження, збереження, обробки та отримання доступу до файлів при розробці веб сервісу. Використано технологію Azure Blob storage в якості сховища даних та розроблено програмний модуль для інтеграції з даним сховищем для подальшого використання в сервісах системи.
5. В ході розробки системи було використано шаблони проектування та сучасні принципи ООП для створення гнучкого та швидкого програмного забезпечення із слабкою зв’язністю компонентів. Загальні конфігурації, методи розширення,

контракти та компоненти мікросервісів було винесено в NuGet пакети та розміщено на сервері Azure.

6. Було досліджено та порівняно різні види автентифікації користувачів, а також розроблено middleware для реалізації автентифікації та авторизації на основі сесійного ключа та API ключа.
7. Було досліджено особливості роботи та порівняно SQL та NoSQL бази даних та використано в якості сховищ даних в залежності від особливостей. PostgreSQL в якості SQL сховища було обрано для збереження основних даних сервісу, а MongoDB в якості NoSQL сховища було обрано для даних, які будуть приймати участь в полінгу, через що потребується швидкий доступ до них.

Результати дослідження бакалаврської роботи апробовані на всеукраїнських науково-технічних конференціях: “Застосування програмного забезпечення в ІКТ” та “XIV Науково-технічна конференція студентів та молодих вчених «Сучасні інфокомунікаційні технології»”.

Перелік посилань

1. Price M. J. C# 9 and .NET 5 – Modern Cross-Platform Development: Build intelligent apps, websites, and services with Blazor, ASP.NET Core, and Entity Framework Core using Visual Studio Code, 5th Edition / Mark Price., 2020. – 361-465 с.
2. Lock A. Asp.net Core in Action / Andrew Lock., 2021. – 255-358 с.; 667-744 с.
3. Петрик М. Моделювання програмного забезпечення / М. Петрик, О. Петрик. "– Тернопіль: Видавництво Тернопільського національного технічного університету імені Івана Пулюя, 2015." – 20-172 с.
4. Shvets A. Dive into design patterns / Alexandr Shvets., 2021. – 32-71 с.; 105-124 с.; 250-268 с.
5. Horsdal C. Microservices in .NET Core / Christian Horsdal., 2017. – 55-248 с.
6. Microservices vs Monolith. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/ Дата звернення: 11.04.2022
7. Overview to ASP.NET Core. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-5.0 Дата звернення: 11.04.2022
8. Relational vs. NoSQL data. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/relational-vs-nosql-data Дата звернення: 11.04.2022
9. What is PostgreSQL. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://www.postgresqltutorial.com/postgresql-getting-started/what-is-postgresql/ Дата звернення: 11.04.2022
10. What is MongoDB – Working and Features. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/what-is-mongodb-working-and-features/ Дата звернення: 11.04.2022
11. A Comparative Study: MongoDB vs. MySQL. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://www.researchgate.net/publication/278302676_A_Comparative_Study_MongoDB_vs_MySQL Дата звернення: 11.04.2022
12. What is Message Broker. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://ademcatamak.medium.com/what-is-message-broker-4f6698c73089 Дата звернення: 11.04.2022

13. Message Brokers. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://www.ibm.com/nl-en/cloud/learn/message-brokers Дата звернення: 11.04.2022
14. Introduction to Azure Blob storage. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction Дата звернення: 11.04.2022
15. What is Docker. [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://aws.amazon.com/docker/ Дата звернення: 11.04.2022
16. Code First vs Database First vs Model First – EntityFramework Approaches Explained. (“Code First vs Database First vs Model First ...”) [Електронний ресурс]: [Веб-сайт]. – електронні дані. – Режим доступу до ресурсу: https://dotnetcoretutorials.com/2021/06/26/code-first-vs-database-first-vs-model-first-entityframework-approaches-explained/ Дата звернення: 11.04.2022

ДОДАТОК А
Приклади з коду

```
[HttpGet]
[RequiresMembership]
simple enough (5%)  Alexandr Kirpichyov
public async Task<GroupSummaryModel> GetSummary()
{
    GroupSummaryDto summaryDto = await _groupsService.GetGroupSummary();
    return _mapper.Map<GroupSummaryDto, GroupSummaryModel>(summaryDto);
}
```

Рисунок 1. – Мапинг з DTO в Model

```
namespace AssistEasy.Application.Mapping
{
    simple enough (5%)  Alexandr Kirpichyov
    public class GroupsProfile : Profile
    {
        simple enough (5%)  Alexandr Kirpichyov
        public GroupsProfile()
        {
            CreateMap<Group, GroupSummaryDto>();
            CreateMap<GroupMember, GroupMemberDto>()
                .ForMember(dto => dto.Username, rule => rule.MapFrom(dto => dto.User.Username))
                .ForMember(dto => dto.Fullname, rule => rule.MapFrom(dto => dto.User.Fullname))
                .ForMember(dto => dto.AvatarUrl, rule => rule.MapFrom(dto => dto.User.AvatarUrl));

            CreateMap<CreateGroupDto, Group>();
            CreateMap<GroupRoleDto, GroupRole>();

            CreateMap<GroupRole, UpdatedRole>()
                .ConvertUsingEnumMapping(opt => opt.MapByName());

            CreateMap<GroupPage, GroupPageDto>();
        }
    }
}
```

Рисунок 2. – Правила мапингу для моделей групи

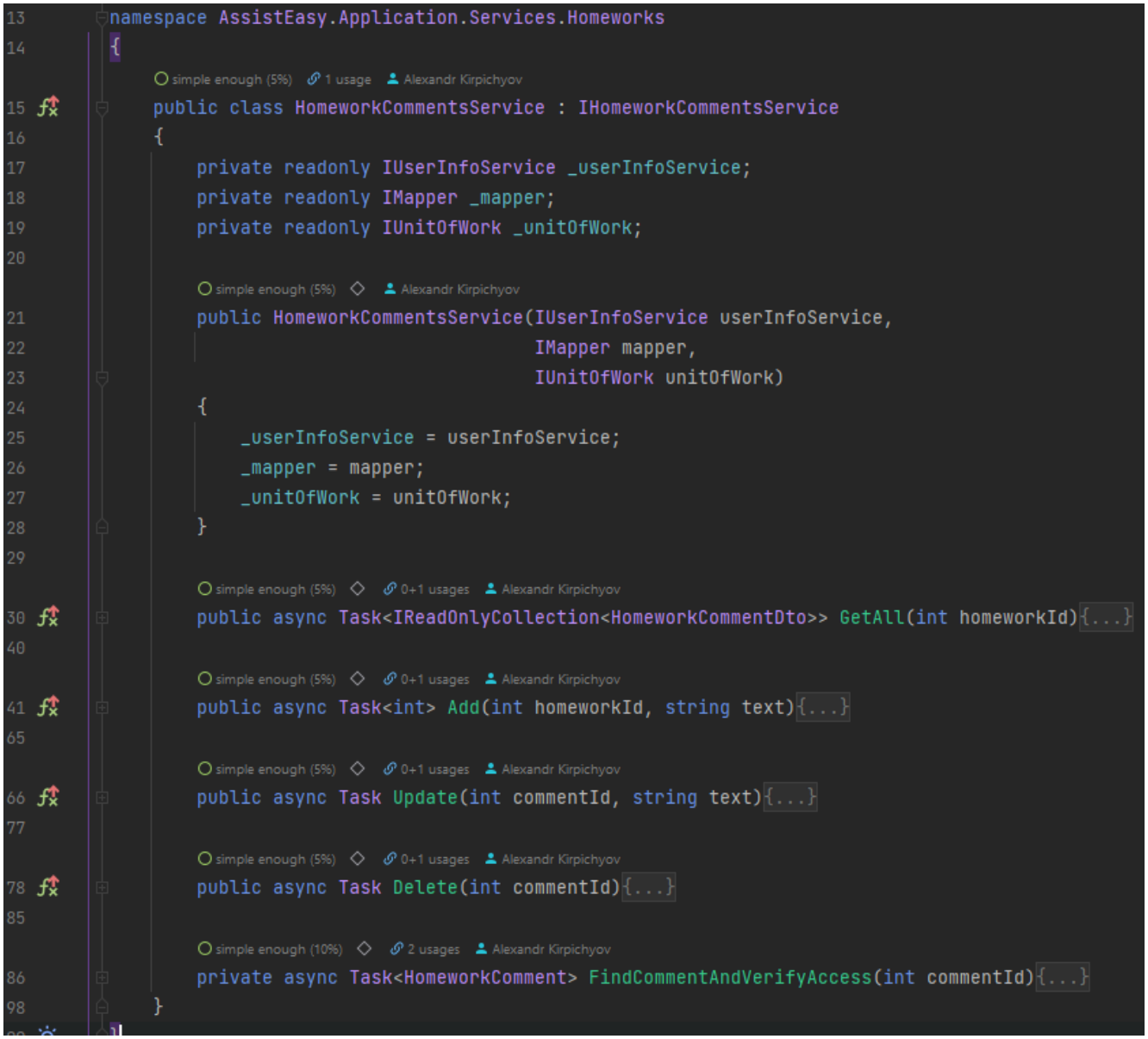


Рисунок 3. – Сервіс, що має єдину відповідальність

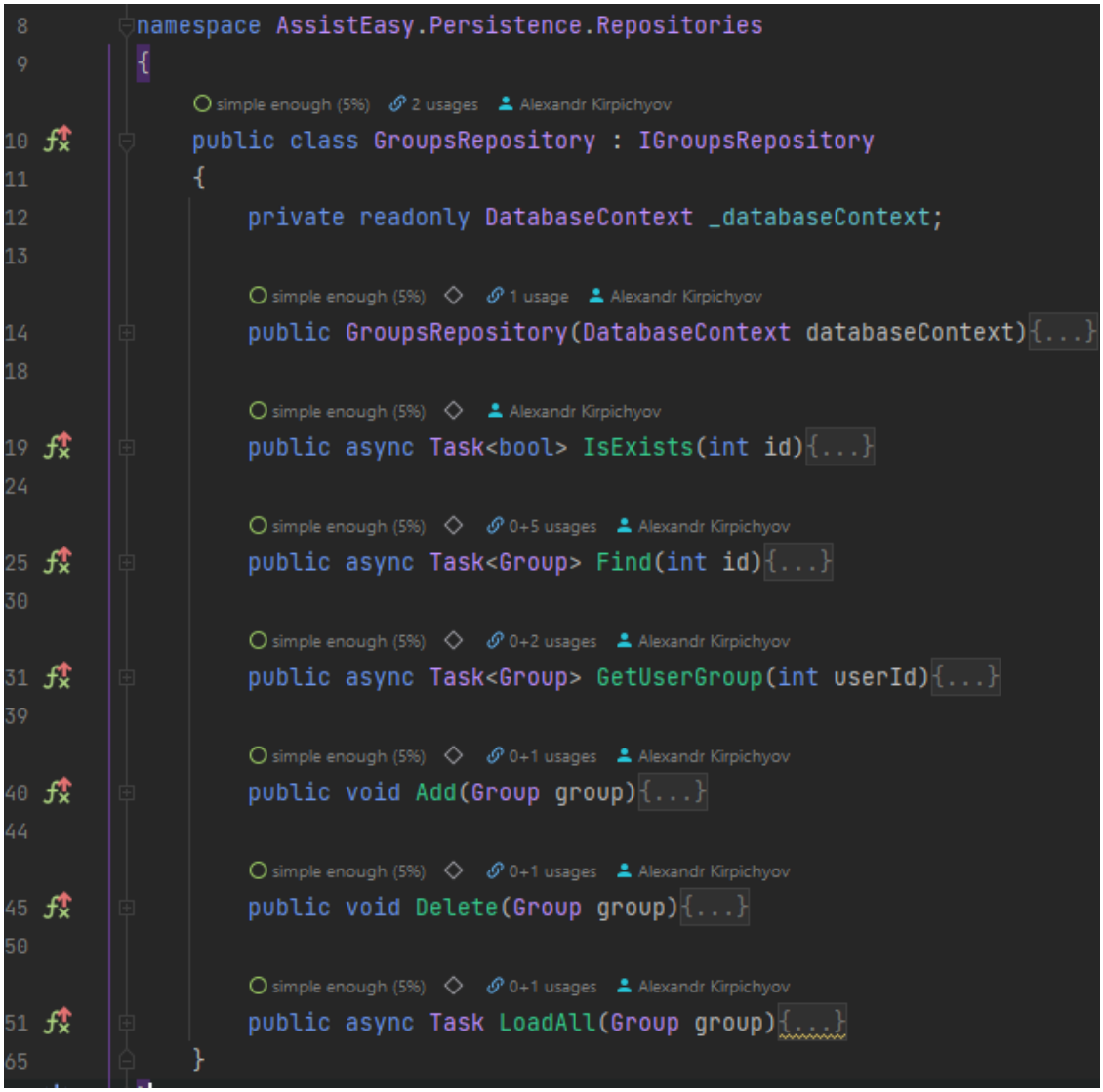


Рисунок 4. – Сервіс, що має єдину відповідальність

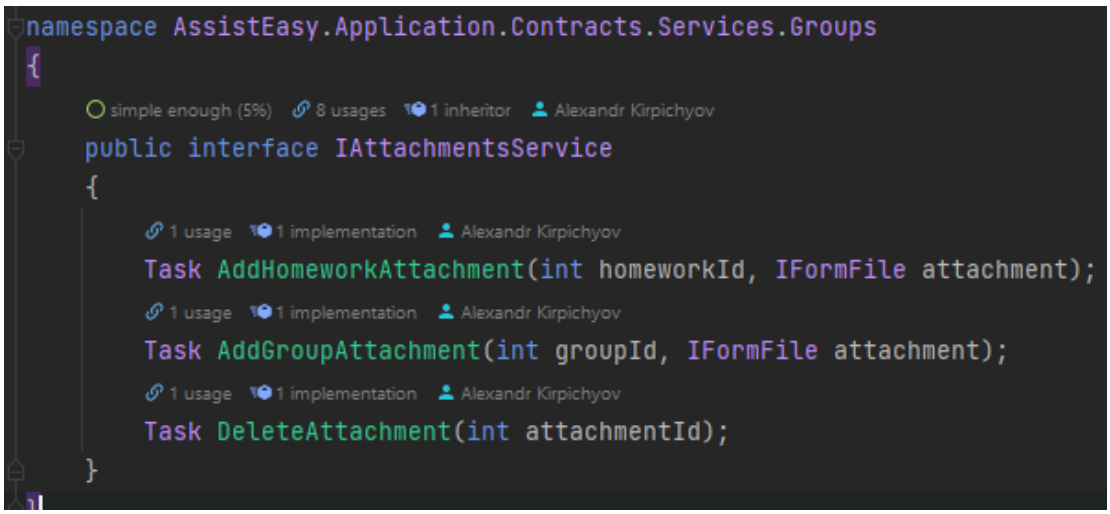


Рисунок 5. – Інтерфейс для сервісу вкладень

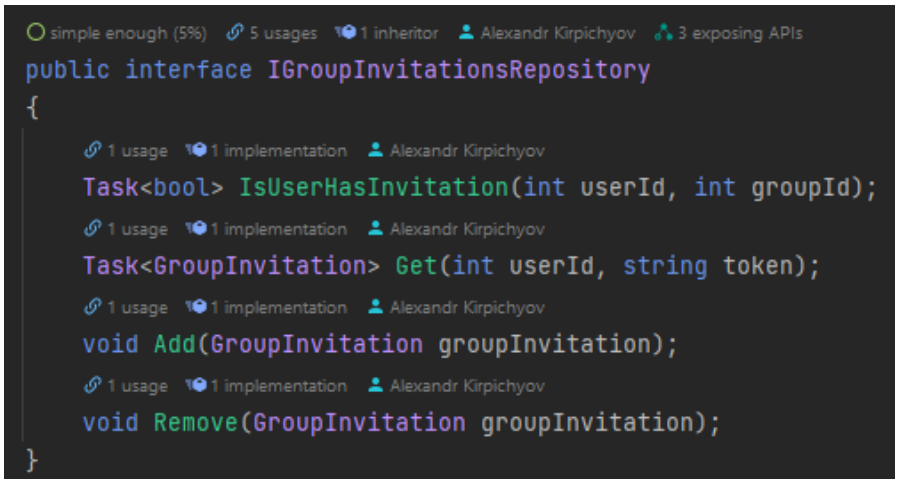


Рисунок 6. – Інтерфейс для репозиторію

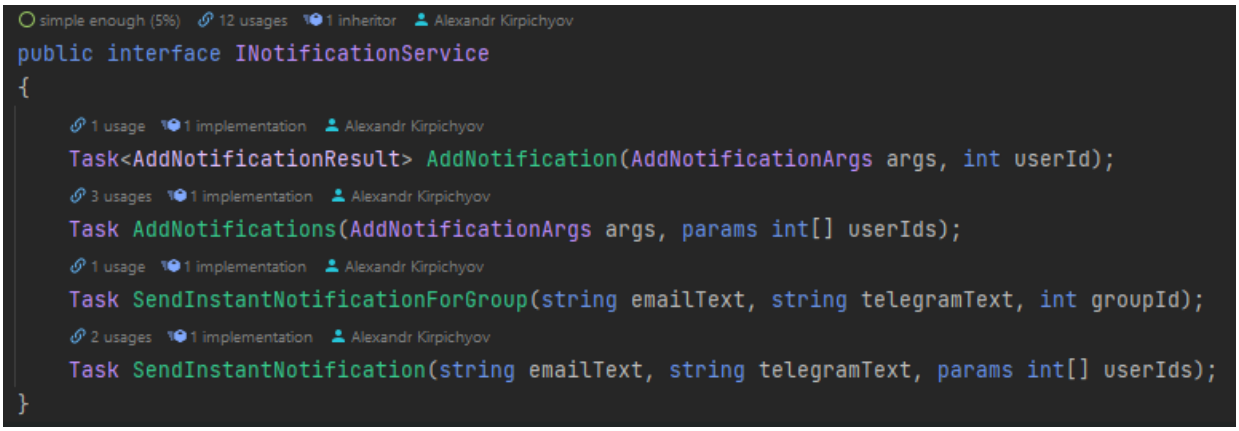
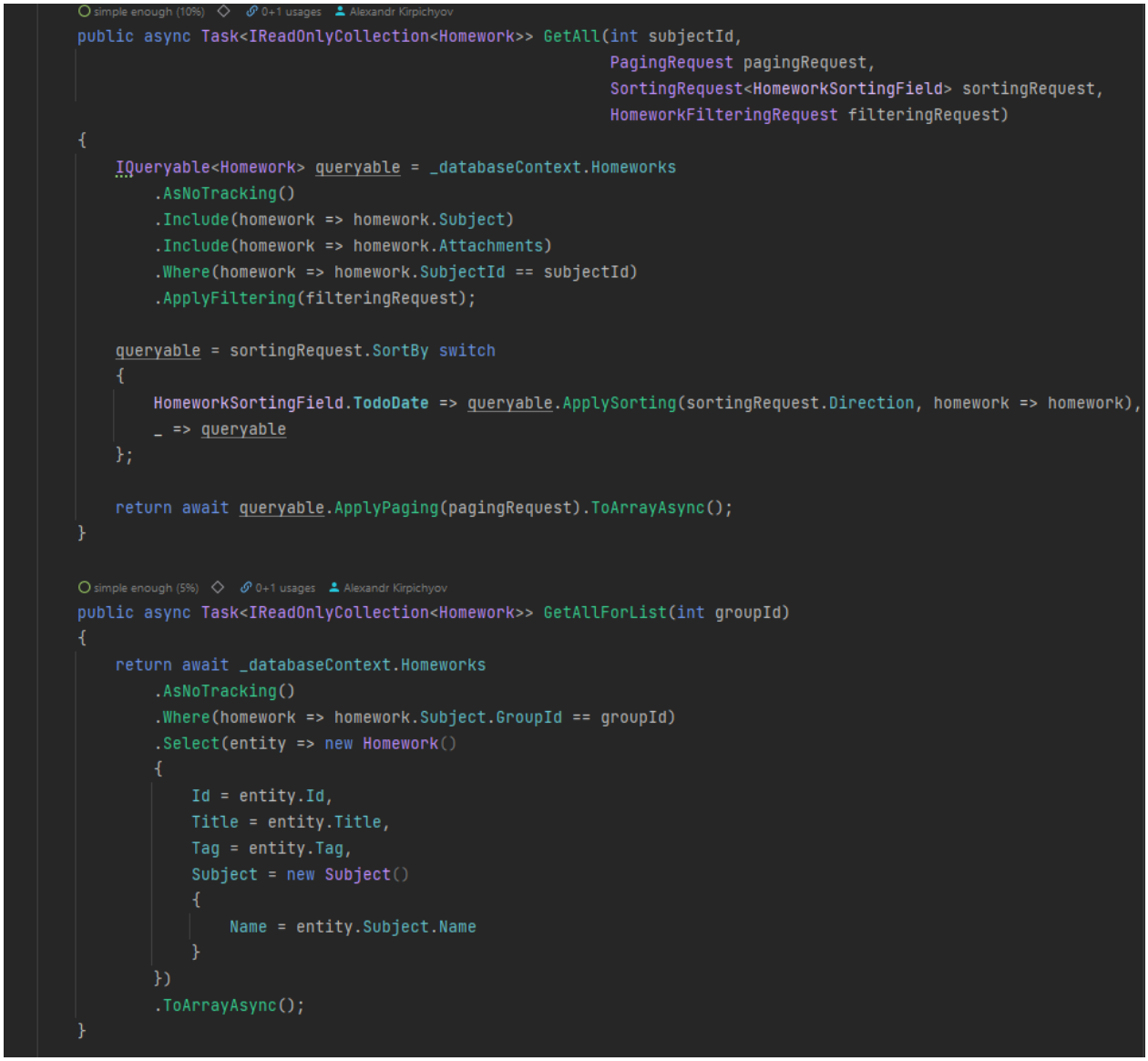


Рисунок 7. – Інтерфейс для сервісу повідомлень

Рисунок 8. – Конструктор сервісу

Рисунок 9. – Одиниця роботи

Рисунок 10. – Контекст бази даних EFCore



```
simple enough (10%)  0+1 usages  Alexandr Kirpichyov
public async Task<IReadOnlyCollection<Homework>> GetAll(int subjectId,
                                                        PagingRequest pagingRequest,
                                                        SortingRequest<HomeworkSortingField> sortingRequest,
                                                        HomeworkFilteringRequest filteringRequest)
{
    IQueryable<Homework> queryable = _databaseContext.Homeworks
        .AsNoTracking()
        .Include(homework => homework.Subject)
        .Include(homework => homework.Attachments)
        .Where(homework => homework.SubjectId == subjectId)
        .ApplyFiltering(filteringRequest);

    queryable = sortingRequest.SortBy switch
    {
        HomeworkSortingField.TodoDate => queryable.ApplySorting(sortingRequest.Direction, homework => homework),
        _ => queryable
    };

    return await queryable.ApplyPaging(pagingRequest).ToArrayAsync();
}

simple enough (5%)  0+1 usages  Alexandr Kirpichyov
public async Task<IReadOnlyCollection<Homework>> GetAllForList(int groupId)
{
    return await _databaseContext.Homeworks
        .AsNoTracking()
        .Where(homework => homework.Subject.GroupId == groupId)
        .Select(entity => new Homework()
        {
            Id = entity.Id,
            Title = entity.Title,
            Tag = entity.Tag,
            Subject = new Subject()
            {
                Name = entity.Subject.Name
            }
        })
        .ToArrayAsync();
}
```

Рисунок 11. – Приклад виконання запитів EFCore (читання)

```
simple enough (15%)  0+1 usages  Alexandr Kirpichyov
public async Task<int> Create(CreateHomeworkDto createDto)
{
    Subject existingSubject = await _unitOfWork.Subjects.Get(createDto.SubjectId);

    if (existingSubject == null)
    {
        throw new NotFoundException(ApiErrorCodes.NotFound(nameof(Subject)));
    }

    if (_userInfoService.GroupId != existingSubject.GroupId)
    {
        throw new UnauthorizedAccessException();
    }

    Homework newHomework = _mapper.Map<CreateHomeworkDto, Homework>(createDto);
    newHomework.UploadDate = DateTime.UtcNow;

    _unitOfWork.Homeworks.Add(newHomework);
    await _unitOfWork.SaveChangesAsync();
```

Рисунок 12. – Приклад виконання запитів EFCore (створення)

```
simple enough (5%)  1 usage  Alexandr Kirpichyov
public class MongoDbContext : IMongoDbContext
{
    private readonly MongoClient _mongoClient;
    private readonly IMongoDatabase _database;

    1+1 usages
    public IMongoCollection<UserNotification> UserNotifications { get; }
    1+1 usages
    public IMongoCollection<OnlineClass> OnlineClasses { get; }
    1+1 usages
    public IMongoCollection<AuthSession> AuthSessions { get; }

    simple enough (5%)  Alexandr Kirpichyov
    public MongoDbContext(IOptions<MongoDbOptions> mongoDbOptions)
    {
        _mongoClient = new MongoClient(mongoDbOptions.Value.ConnectionUrl);
        _database = _mongoClient.GetDatabase(mongoDbOptions.Value.DatabaseName);

        UserNotifications = _database.GetCollection<UserNotification>(mongoDbOptions.Value.UserNotificationsCollectionName);
        OnlineClasses = _database.GetCollection<OnlineClass>(mongoDbOptions.Value.OnlineClassesCollectionName);
        AuthSessions = _database.GetCollection<AuthSession>(mongoDbOptions.Value.AuthSessionsCollectionName);
    }
}
```

Рисунок 13. – Використання контексту для підключення до MongoDB

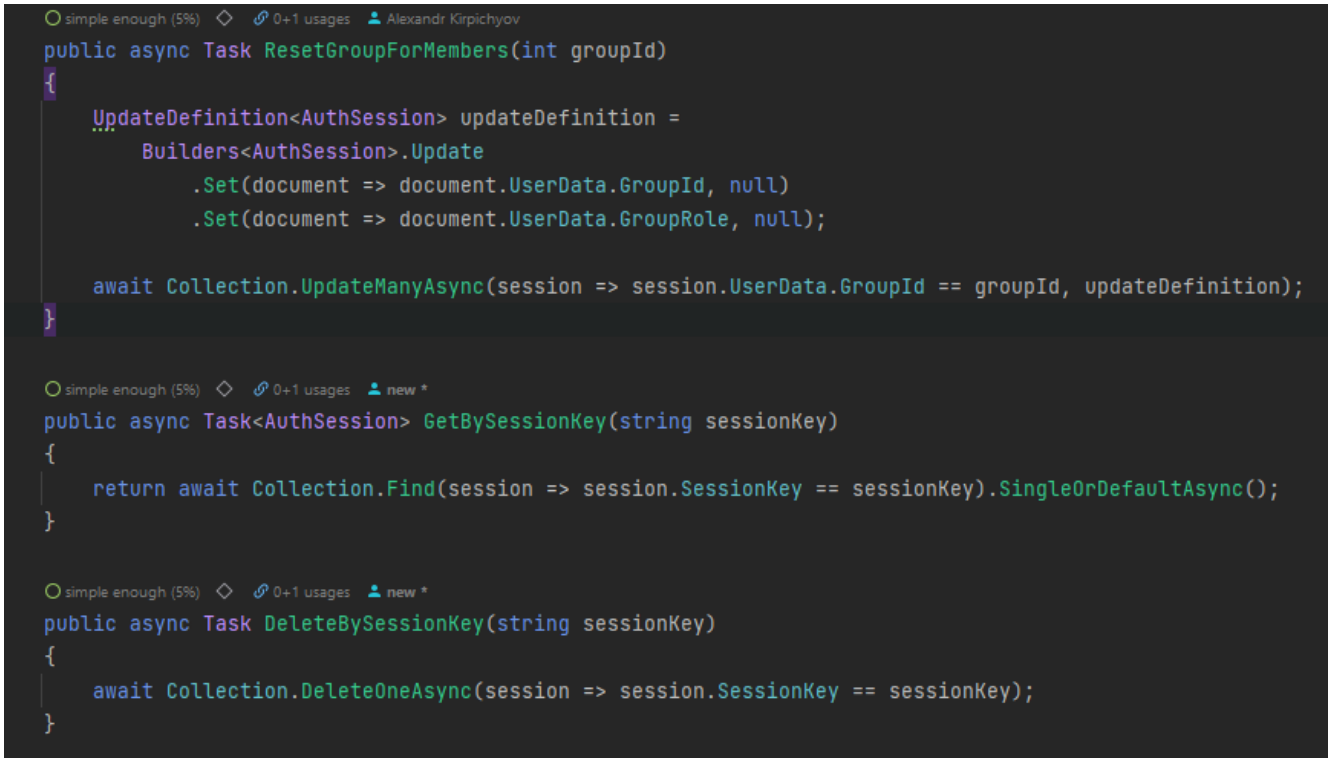


Рисунок 14. – Приклад виконання запитів MongoDB.Driver

```
[DateBasedMigration(year: 2021, month: 10, day: 02, hhmm: 0154)]
simple enough (5%)  Alexandr Kirpichyov
public class AddHomeworkFeedbacksTable : Migration
{
    simple enough (5%)  Alexandr Kirpichyov
    public override void Up()
    {
        Create.Table(TableNames.HomeworkFeedbacks)
            .WithColumn("id").AsInt32().PrimaryKey().Identity()
            .WithColumn("homework_id")
                .AsInt32()
                .ForeignKey(TableNames.Homeworks, "id")
                .OnDelete(Rule.Cascade)
                .NotNullable()
            .WithColumn("group_member_id")
                .AsInt32()
                .ForeignKey(TableNames.GroupMembers, "id")
                .OnDelete(Rule.Cascade)
                .NotNullable()
            .WithColumn("difficulty_score").AsInt32().NotNullable()
            .WithColumn("takes_time_minutes").AsInt32().NotNullable()
            .WithColumn("updated_at_utc").AsDateTime().NotNullable();
    }

    simple enough (5%)  Alexandr Kirpichyov
    public override void Down()
    {
        Delete.Table(TableNames.HomeworkFeedbacks);
    }
}
```

Рисунок 15. – Приклад міграції з використанням FluentMigration

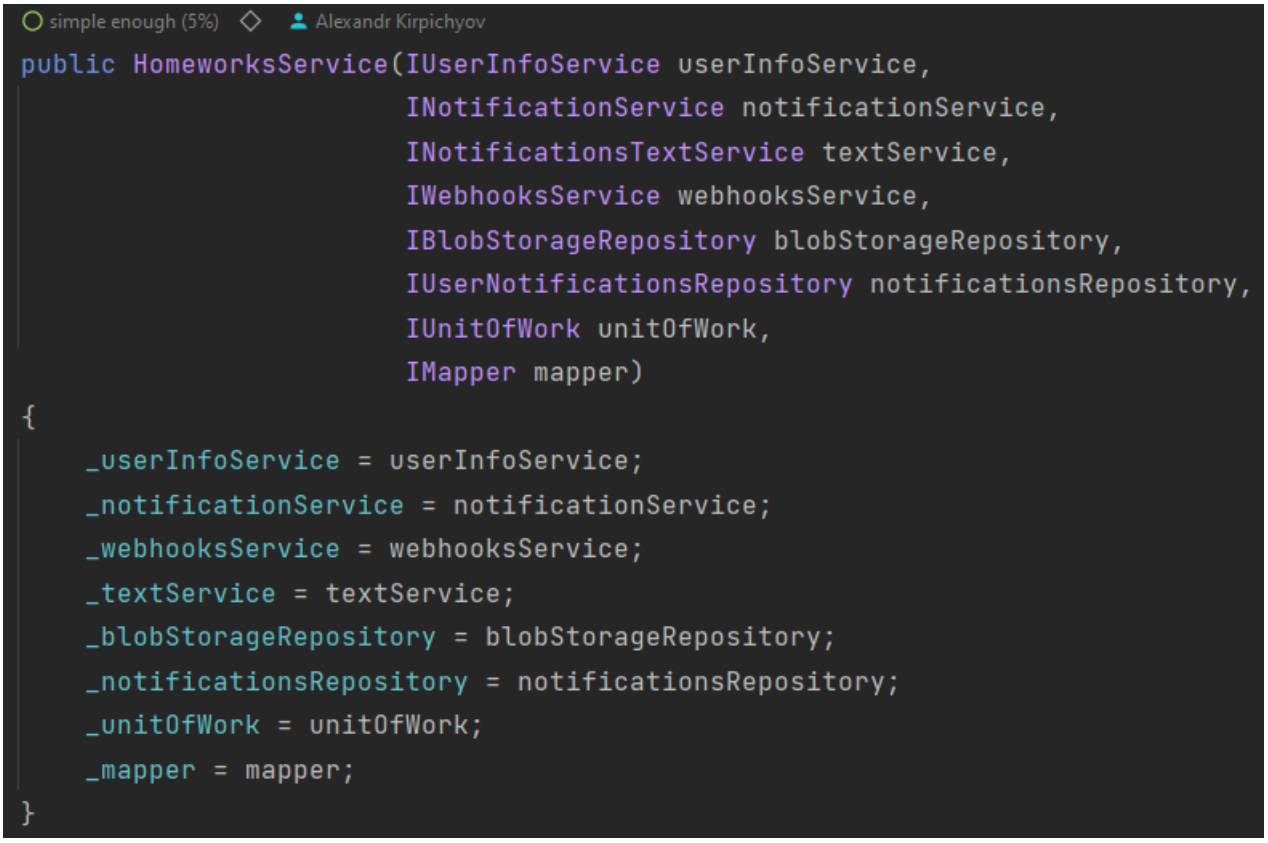


Рисунок 16. – Використання DI - конструктор сервісу

```
simple enough (5%)   1 usage   Alexandr Kirpichyov +2
public static IServiceCollection AddApplicationServices(this IServiceCollection services)
{
    services.AddScoped<IUserInfoService, UserInfoService>();
    services.AddScoped<ITokensGenerator, TokensGenerator>();
    services.AddScoped<IIdentityService, IdentityService>();
    services.AddScoped<IConfirmationTokensService, ConfirmationTokensService>();
    services.AddScoped<IUserProfileService, UserProfileService>();
    services.AddScoped<IGroupsService, GroupsService>();
    services.AddScoped<ISubjectsService, SubjectsService>();
    services.AddScoped<IHomeworksService, HomeworksService>();
    services.AddScoped<IStudentHomeworksService, StudentHomeworksService>();
    services.AddScoped<INotificationService, NotificationsService>();
    services.AddScoped<IOnlineClassesService, OnlineClassesService>();
    services.AddScoped<IHomeworkCommentsService, HomeworkCommentsService>();
    services.AddScoped<IHomeworkFeedbacksService, HomeworkFeedbacksService>();
    services.AddScoped<IAttachmentsService, AttachmentsService>();
    services.AddScoped<IGroupPagesService, GroupPagesService>();
    services.AddScoped<IIntegrationService, IntegrationService>();
    services.AddScoped<IWebhooksService, WebhooksService>();

    services.AddSingleton<INotificationsTextService, NotificationsTextService>();

    return services;
}
```

Рисунок 17. – Використання DI - конфігурування контейнеру залежностей

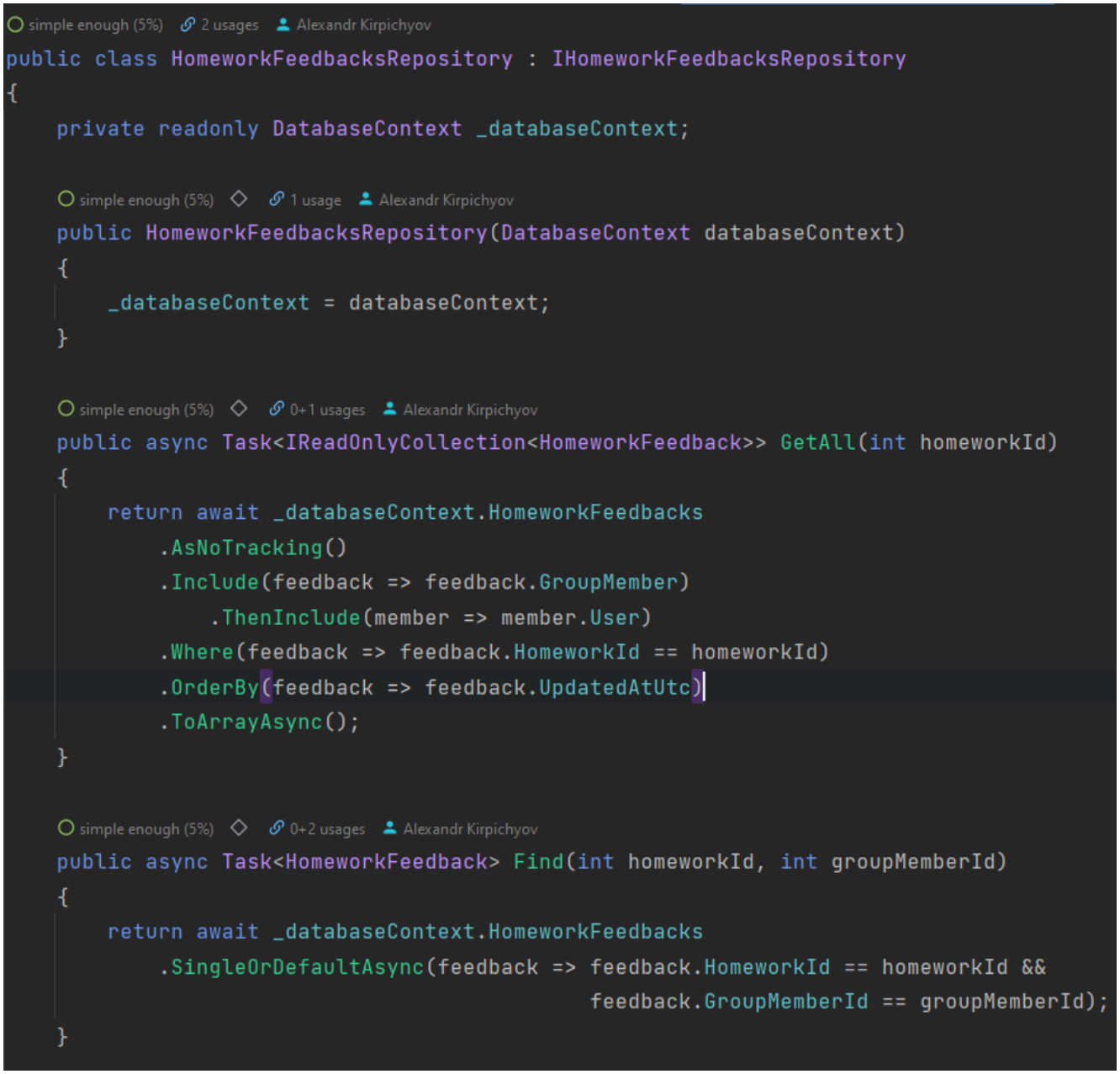


Рисунок 18. – Використання репозиторію в AssistEasy

Рисунок 19. – Використання одиниці роботи в AssistEasy

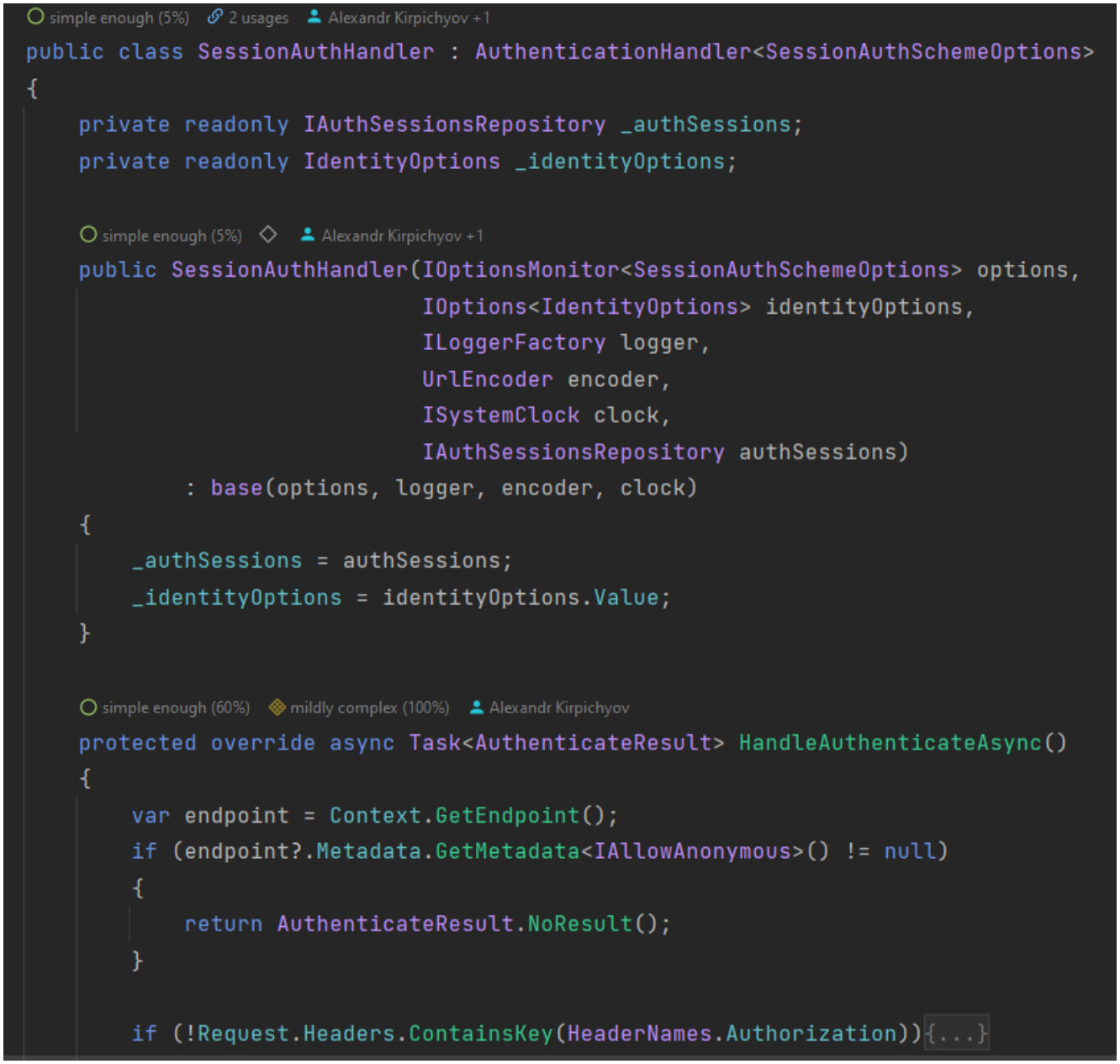


Рисунок 20. – Middleware для автентифікації

Рисунок 21. – Action фільтр для обробки виключень

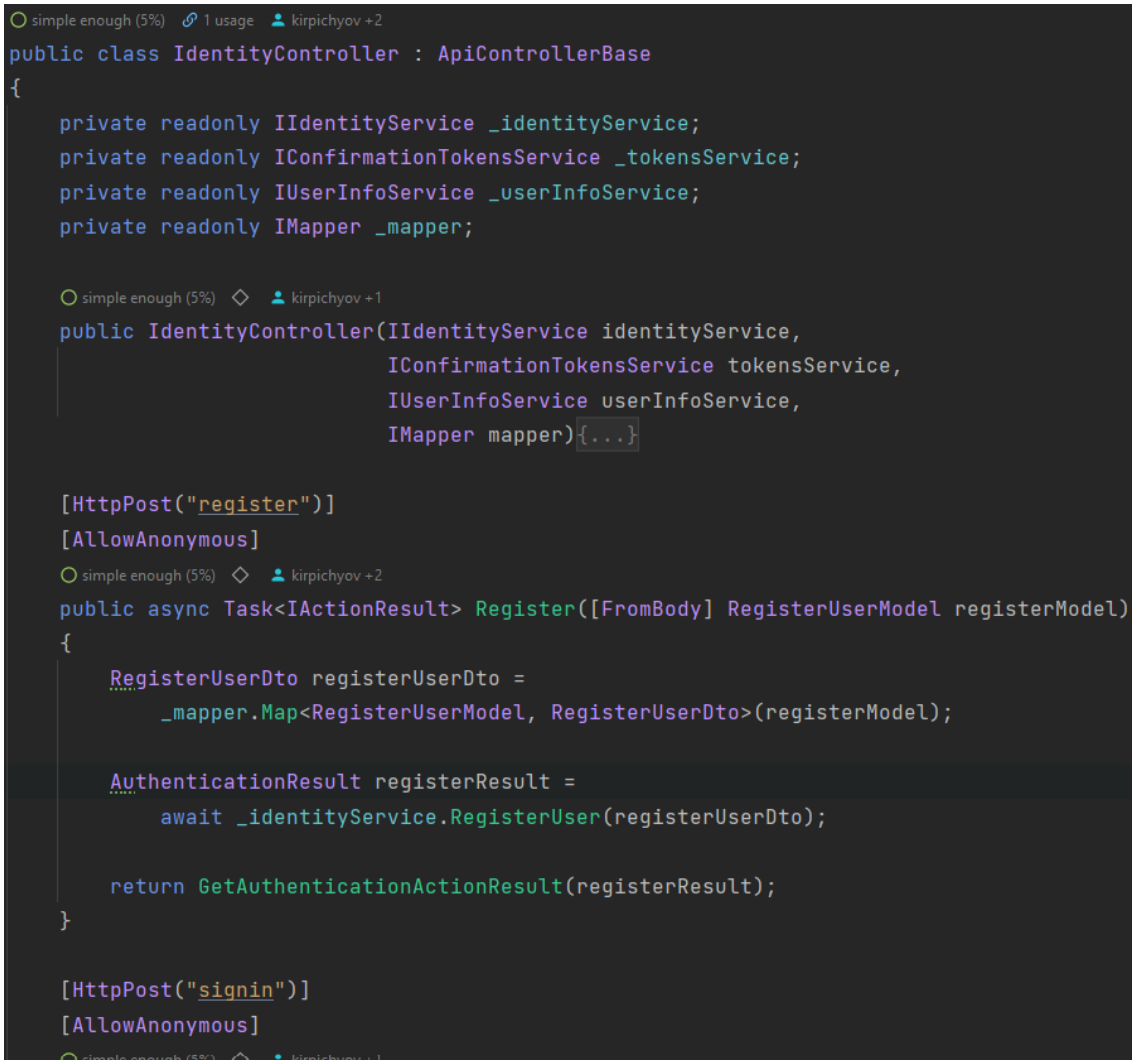


Рисунок 22. – Приклад використання MVC (controller)

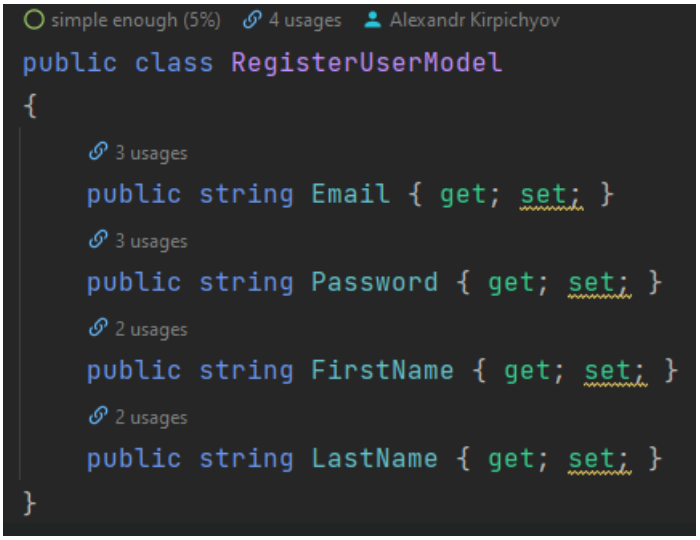


Рисунок 23. – Приклад використання MVC (model)

Server response

Code	Details
200	Response body


```
{
  "success": true,
  "sessionKey": "IPzJvsE2csLG96kKX9u51FaG28oALhPHL0EAxXQKWGA6SjUKQs0wR1pCysQwDDq4",
  "expiresAtUtc": "2022-06-28T10:20:10.9348563Z",
  "errors": []
}
```


Рисунок 24. – Приклад використання MVC (відповідь JSON - view)

```
simple enough (5%)  1 usage  Alexandr Kirpichyov +1
private static IHostBuilder CreateHostBuilder(string[] args) =>
    HostExtensions.CreateDefaultBuilder(args)
                  .UseDefaultServiceProvider((context, options) =>
                  {
                      options.ValidateScopes = true;
                      options.ValidateOnBuild = true;
                  })
                  .UseSerilog()
                  .ConfigureWebHostDefaults(webBuilder =>
                  {
                      webBuilder.UseStartup<Startup>();
                  });
```

Рисунок 25. – Приклад використання будівельника в ASP.NET

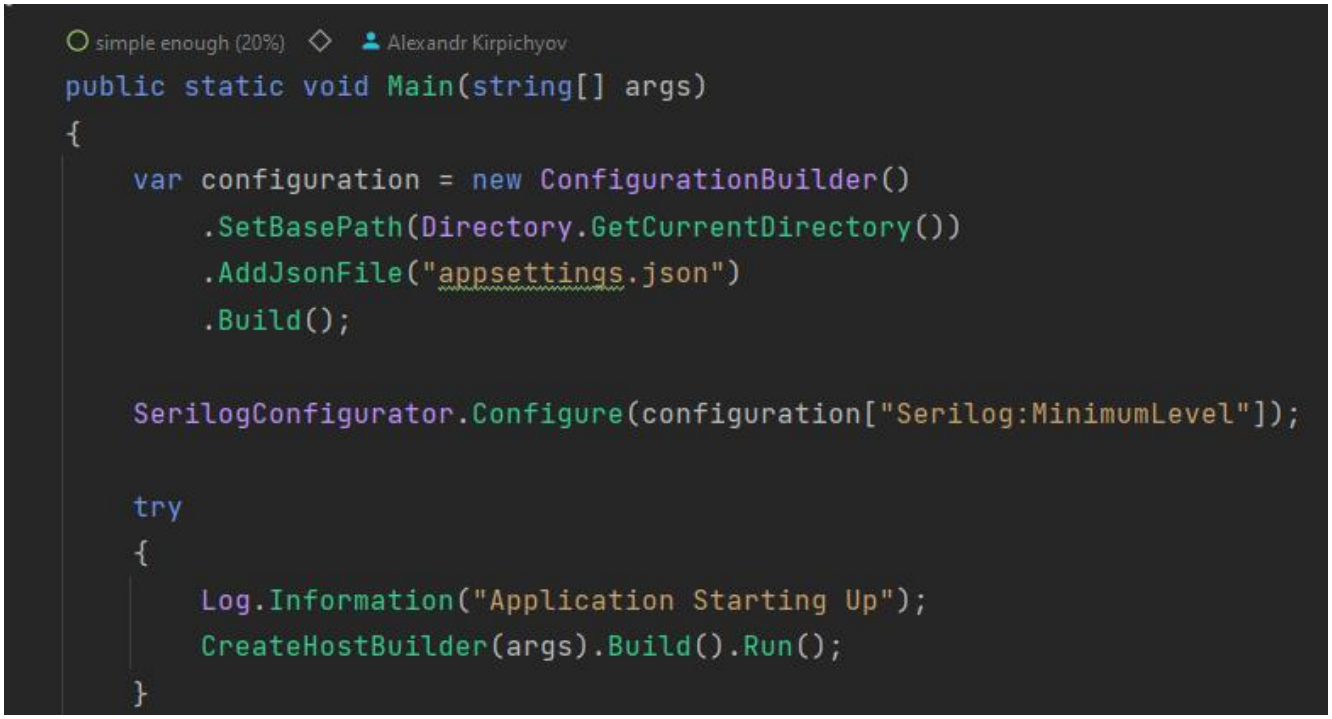


Рисунок 26. – Приклад використання будівельника в ASP.NET

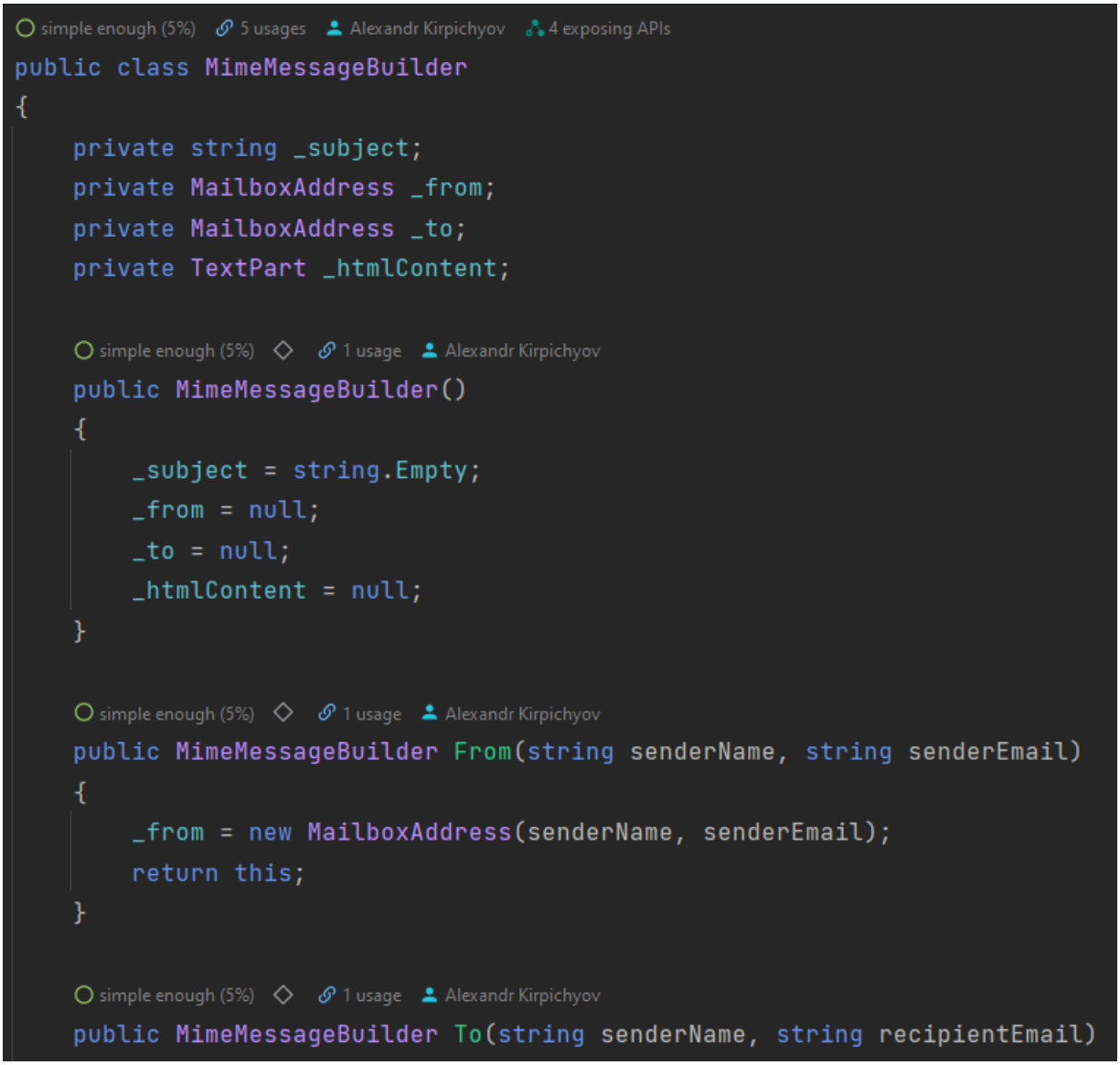


Рисунок 27. – Приклад використання будівельника в AssistEasy.MailSender

```
MimeMessage emailMessage = new MimeMessageBuilder()
    .From(_mailingOptions.FromName, _mailingOptions.Email)
    .To(string.Empty, sendEmailArgs.Recipient)
    .WithSubject(sendEmailArgs.Subject)
    .WithHtmlContent(emailContent)
    .Build();

await SendMessage(emailMessage);
```

Рисунок 28. – Приклад використання будівельника в AssistEasy.MailSender

```
services.AddSwaggerGenNewtonsoftSupport()
        .AddSwaggerGen(options =>
        {
            options.SwaggerDoc("v1", new OpenApiInfo
            {
                Version = "v1",
                Title = "AssistEasy API"
            });

            options.AddSecurityDefinition("SessionKey", new OpenApiSecurityScheme
            {
                Name = HeaderNames.Authorization,
                Type = SecuritySchemeType.ApiKey,
                Scheme = "Basic",
                In = ParameterLocation.Header,
                Description = "Obtained unique SessionKey."
            });

            options.OperationFilter<AuthOperationFilter>();
        });
```

Рисунок 29. – Підключення Swagger до AssistEasy

```
[HttpPost("onlineclasses")]
[RequireRoles(GroupRole.Owner, GroupRole.Admin)]
[ProducesResponseType(typeof(CreatedIdModel<string>), StatusCodes.Status201Created)]
[ProducesResponseType(typeof(BadRequestModel), StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status403Forbidden)]
simple enough (5%)   Alexandr Kirpichyov *
public async Task<IActionResult> Add([FromBody] CreateOnlineClassModel classModel)
{
    CreateOnlineClassDto createOnlineClassDto =
        _mapper.Map<CreateOnlineClassModel, CreateOnlineClassDto>(classModel);

    string id = await _onlineClassesService.Add(createOnlineClassDto);

    return CreatedWithId(id);
}
```

Рисунок 30. – Атрибути Swagger в контролерах AssistEasy

```
[Fact]
simple enough (5%)  1 test OK  Alexandr Kirpichyov
public async Task RegisterUser_EmailIsNotExists_ShouldBeEquivalentToExpected()
{
    // Arrange
    var model = new RegisterUserDto()
    {
        Email = _faker.Person.Email,
        Password = _faker.Internet.Password(),
        FirstName = _faker.Person.FirstName,
        LastName = _faker.Person.LastName
    };

    string sessionKey = _faker.Random.AlphaNumeric(_options.Value.SessionKeyLength);
    DateTime expiresAt = DateTime.UtcNow.AddDays(_options.Value.SessionDaysLifetime);

    var httpContext = new DefaultHttpContext()
    {
        Connection =
        {
            RemoteIpAddress = _faker.Internet.IpAddress()
        }
    };

    _unitOfWorkFakeWrapper.Users
        .CallsTo(repository => repository.IsEmailExist(model.Email))
        .Returns(false);

    _httpContextAccessorFake.CallsTo(context => context.HttpContext)
```

Рисунок 31. – Unit тест для методу IdentityService

```
_httpContextAccessorFake.CallsTo(context => context.HttpContext)
                        .Returns(httpContext);

_tokensGeneratorFake.CallsTo(generator => generator.GenerateTokenAsCode(_options.Value.SessionKeyLength))
                    .Returns(sessionKey);

var sut = BuildSut();

// Act
var result = await sut.RegisterUser(model);

// Assert
using (new AssertionScope())
{
    result.Errors.Should().BeNullOrEmpty();
    result.IsSuccess.Should().BeTrue();
    result.SessionKey.Should().Be(sessionKey);
    result.ExpiresAtUtc.Should().NotBeNull();
    result.ExpiresAtUtc.IsNearlyEqual(expiresAt, TimeSpan.FromSeconds(5)).Should().BeTrue();

    _unitOfWorkFakeWrapper.Users
        .CallsTo(repository => repository.AddUser(A<User>._))
        .MustHaveHappenedOnceExactly();

    _authSessionRepository
        .CallsTo(repository => repository.Insert(A<AuthSession>._))
        .MustHaveHappenedOnceExactly();
}
```

Рисунок 32. – Unit тест для методу IdentityService

ДОДАТОК Б
Демонстраційні матеріали

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ВЕБ-СЕРВІС АСИСТЕНТ ДЛЯ СТУДЕНТІВ ТА УЧНІВ
“ASSIST EASY” МОВОЮ C#»

Виконав студент 4 курсу
групи ПД-43
Кирпичов О.П.
Керівник роботи
Поперешняк С.В.

Київ - 2022

Мета, об'єкт та предмет дослідження

- **Мета роботи** – спрощення та підвищення ефективності процесу навчання для студентів та учнів за допомогою веб -сервісу та сучасних прикладних програмних інтерфейсів

- **Об'єкт дослідження** – здійснення навчання за допомогою інформаційних технологій

- **Предмет дослідження** – веб-сервіс асистент для студентів та учнів на основі мікросервісів

Актуальність роботи

• Пандемія COVID-19 призвела до переведення значної кількості навчальних закладів на дистанційне навчання, але учбові заклади та учні не були готові до такої різкої зміни в навчальному процесі. Через неготовність до таких подій багато закладів не мали уніфікованого плану дій щодо організації дистанційних занять, а на студентів звалилося набагато більше матеріалу для опрацювання та завдань у порівнянні з очним навчанням.

• Стало достатньо складно тримати все в голові та слідкувати за змінами, дедлайнами, а також посиланнями та розкладом онлайн занять, поширювати нову інформацію між студентами групи або класу, а також ділитися інформацією, порадами та питаннями з предметів між студентами.

• Для вирішення вищезгаданих проблем потрібно мати комплексне рішення на основі інформаційних систем, які зможуть забезпечити швидку та зручну обробку даних, потрібних студентам та учням.

Аналоги

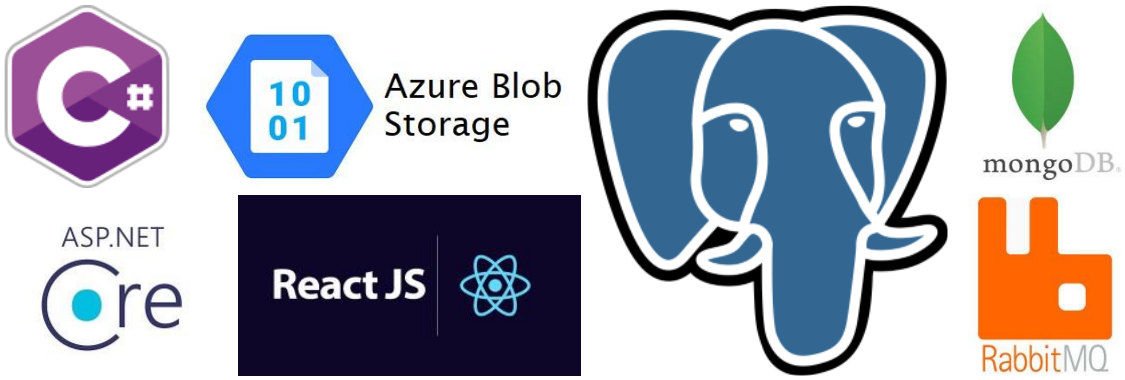
Порівняння з аналогами

	Google Classroom	Moodle	Trello	Assist Easy
Відкритий початковий код	-	+	-	+
Відкритий API	+	*	+	+
Панель керування викладача	+	+	-	-
Панель керування учня	+	+	+	+
Панель керування адміністратора	-	+	*	+
Можливість створення та редагування завдань	+	+	+	+
Можливість прикріплення файлових вкладень	+	+	+	+
Можливість коментування завдань	+	+	+	+
Персональна статистика	*	*	-	+
Групова статистика	-	-	-	+
Можливість перегляду оцінок учнем	+	+	-	-
Можливість отримання сповіщень через Email	+	*	+	+
Можливість отримання сповіщень через Telegram	-	-	-	+
Підтримка користувацьких вебхуків	-	-	+	+
Динамічна головна сторінка групи	-	-	-	+
Підтримка розкладу онлайн занять	-	-	-	+
Користувацький TODO лист	*	*	+	+
Можливість самостійного створення облікового запису	+	-	+	+
Керування власним акаунтом	+	-	+	+

Технічне завдання

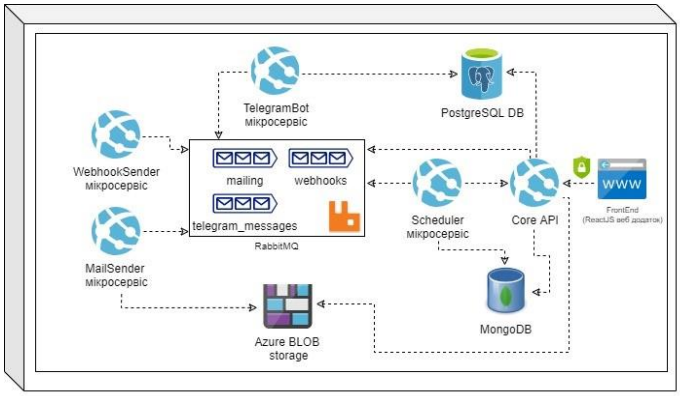
1. Визначити архітектуру системи та її ключові елементи ;
2. Розробити алгоритм створення та доставки користувацьких повідомлень;
3. Дослідити технології для обробки та збереження даних сервісу ;
4. Визначити структуру моделей та баз даних ;
5. Визначити основні схеми аутентифікації та розробити програмні модулі для їх реалізації ;
6. Розробити веб -сервіс асистент на основі мікросервісів для студентів на учнів .

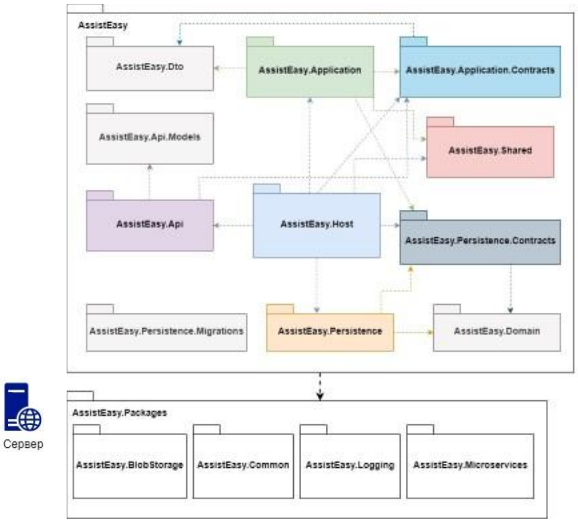
Програмні засоби реалізації



Інструменти використані для реалізації

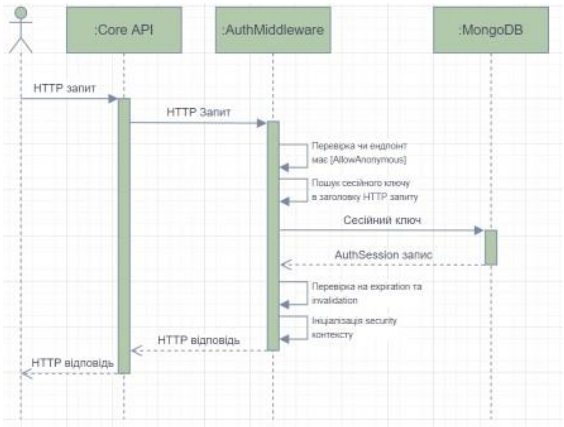
Архітектура сервісу



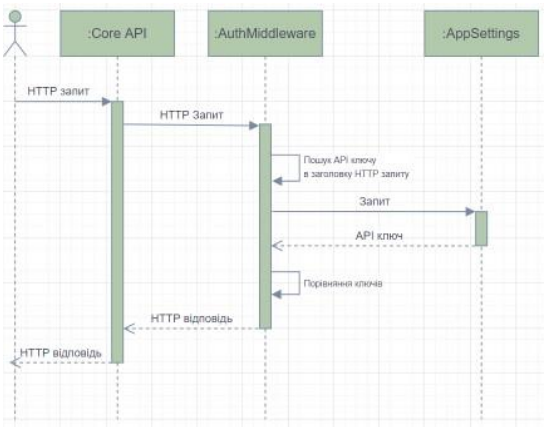


Сервер

Механізм автентифікації

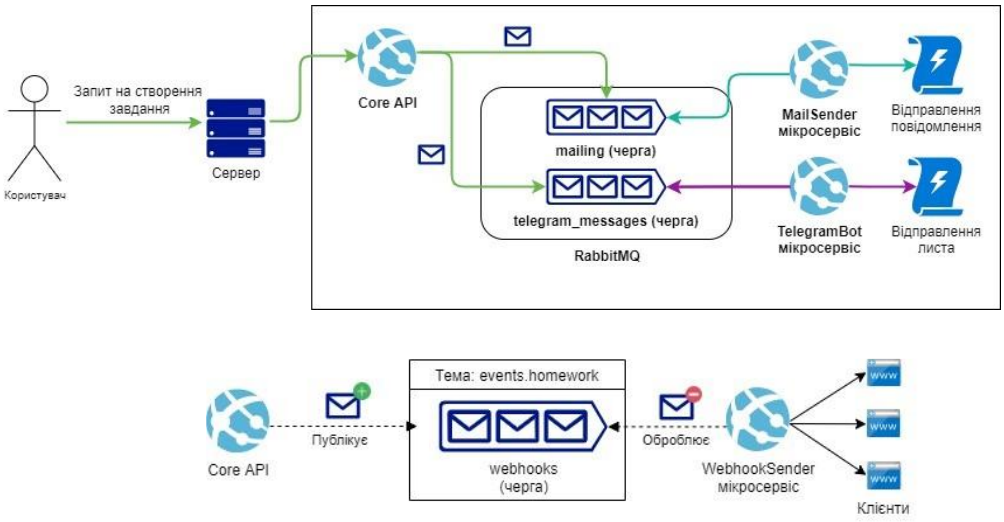


Механізм на основі Session Key

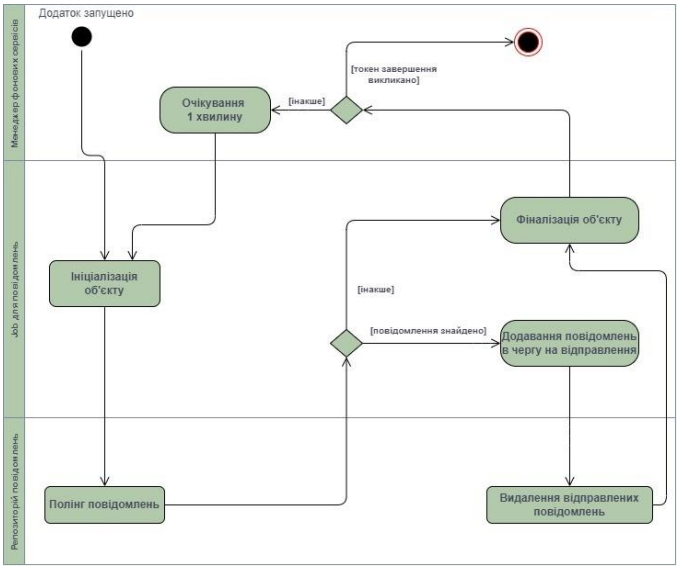


Механізм на основі API Key

Схема відправлення повідомлень



Заплановані сповіщення на основі Job-ів



Апробація результатів дослідження

1. Кирпичов О.П. Розробка веб сервісу асистента для студентів та учнів : Матеріали науково-технічної конференції «Застосування програмного забезпечення в ІКТ». Збірник тез. - 20.04.2022, ДУТ, м.Київ.
2. Кирпичов О.П. Використання SQL та NoSQL баз даних в якості сховищ для сучасного веб сервісу : Матеріали XIV Науково-технічної конференції студентів та молодих вчених «Сучасні інфокомунікаційні технології. Збірник тез. - 19.05.2022, ДУТ, м.Київ.
3. Спроектована архітектура була представлена колегам та висококваліфікованим фахівцям з розробки програмного забезпечення компанії SoftServe рівню Middle та Senior, де отримала схвалення.

Висновки

1. Було обґрунтовано актуальність розробленої системи та її наукову новизну шляхом аналізу близьких за функціоналом продуктів.
2. Було сформовано та описано функціональні та нефункціональні вимоги до системи.
3. Було спроектовано архітектуру системи та розроблено веб сервіс з використанням мікросервісів, сервісів Azure та брокеру повідомлень.
4. Описано програмні засоби та інструменти, за допомогою яких було розроблено програмне рішення.
5. Було досліджено особливості роботи та порівняно SQL та NoSQL бази даних та використано в якості сховищ даних основуючись на їх особливостях.

Перспективи подальших досліджень та розвитку даної роботи

1. Додання можливості підключення та використання інших месенджерів (WhatsApp, Viber, Facebook messenger);
2. Імплементація можливості перегляду оцінок, отриманих студентом;
3. Створення мобільного клієнта ;
4. Розширення персональної та групової статистики ;
5. Можливість мати декілька груп одночасно