Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка програмного забезпечення для керування гуртожитком з використанням фреймворку ASP.NET, Flutter»

Виконав: студент IV курсу
групи ЗПІ-18б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності
Ковтун Богдан Валентинович
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Романюк О.В.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф КН Колодний В.В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____ О.Н. Романюк
«13» червня 2022 р.

ВНТУ – 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
_____
25 березня 2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Ковтуну Богдану Валентиновичу

1. Тема роботи – «Розробка програмного забезпечення для керування гуртожитком з використанням фреймворку ASP.NET, Flutter».

Керівник роботи: Романюк Оксана Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 24 березня 2022 року №66.

2. Строк подання студентом роботи 13 червня 2022 року.

3. Вихідні дані до роботи: Вихідні дані до роботи: модель розробки – Agile; метод передачі повідомлень між серверами – HTTP; вхідні дані – інформація від користувачів, квитанція про оплату; вихідні дані – документи для керування гуртожитком; середовище розробки – VisualStudio; мова програмування – C#; фреймворк – ASP.NET, Flutter; СКБД – MongoDB.

4. Зміст розрахунково-пояснювальної записки: вступ; обґрунтування вибору методу розробки та постановка задачі дослідження; розробка структури, методу та алгоритмів роботи програмного продукту; розробка програмних компонент; тестування додатку; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження; новизна одержаних результатів; практична цінність одержаних результатів; порівняльний аналіз аналогів; структура графічного інтерфейсу головного вікна додатку; структура графічного інтерфейсу вікна сторінки профілю; блок-схема алгоритму створення документів; блок-схема алгоритму зняття коштів з балансу студента; блок-схема

алгоритму нарахування коштів студенту та додавання квитанції в базу да
тестування програми; апробація та публікації результатів роботи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Романюк О.В., к.т.н., доцент кафедри ПЗ	25.03.22	10.06.2

7. Дата видачі завдання 25 березня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примі
1	Аналіз завдання і вибір методу вирішення поставленої задачі дослідження	26.03.2022 – 03.04.2022	Вик
2	Розробка структури графічного інтерфейсу додатку	04.04.2022 – 09.04.2022	Вик
3	Розробка алгоритму створення документів для керування гуртожитком	10.04.2022 – 17.04.2022	Вик
4	Розробка методу та алгоритмів моніторингу боргів студента	18.04.2022 – 28.04.2022	Вик
5	Аналіз і вибір мови програмування та середовища розробки	29.04.2022 - 03.05.2022	Вик.
6	Програмна реалізація додатку	04.05.2022 – 19.05.2022	Вик.
7	Тестування програмного забезпечення	20.05.2022 – 25.05.2022	Вик.
8	Оформлення матеріалів до захисту БДР	26.05.2022 – 10.06.2022	Вик.

Студент _____ **Ковтун Б.**
(підпис) (прізвище та ініціа

Керівник бакалаврської дипломної роботи _____ **Романюк О.**
(підпис) (прізвище та ініціа

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 109 сторінок формату А4, на яких є 46 рисунків, 6 таблиць, список використаних джерел містить 26 найменувань.

У бакалаврській дипломній роботі проведено детальний аналіз аналогів додатків для керування гуртожитком. Сформульовано мету досліджень – підвищення ефективності керування гуртожитком шляхом реалізації функціональних можливостей для моніторингу боргів студента та автоматичної генерації документів.

Запропоновано метод моніторингу боргів студента, шляхом виконання щомісячного автоматичного нарахування боргу та списання боргу при підтвердженні квитанції про сплату студентом, що дозволяє підвищити точність розрахунку боргу студента та спростити процес моніторингу боргів. Удосконалено алгоритм автоматичного генерування документів для керування гуртожитком, який дозволяє завантажувати документ лише після його підпису усіма сторонами, що дало можливість підвищити надійність та достовірність автоматично згенерованих документів.

Програмне забезпечення для керування гуртожитком розроблено з використанням фреймворку ASP.NET, Flutter, мов програмування C#, в середовищі розробки VisualStudio.

Отримані в бакалаврській дипломній роботі результати можна використати в якості допоміжного програмного забезпечення для керування гуртожитком.

Ключові слова: керування гуртожитком, автоматична генерація документів, моніторинг боргів.

ABSTRACT

The bachelor's thesis consists of 109 A4 pages, which contain 46 figures, 6 tables, the list of sources used contains 26 titles.

In the bachelor's thesis a detailed analysis of analogs of applications for dormitory management. The purpose of the research is formulated - to increase the efficiency of dormitory management by implementing functionalities for monitoring student debts and automatic generation of documents.

A method of monitoring student debts is proposed by performing monthly automatic debt accrual and debt write-off upon confirmation of the student's payment receipt, which allows to increase the accuracy of student debt calculation and simplify the debt monitoring process. The algorithm for automatic generation of documents for dormitory management has been improved, which allows downloading a document only after it has been signed by all parties, which has made it possible to increase the reliability and authenticity of automatically generated documents.

The dormitory management software was developed using the ASP.NET framework, Flutter, as a C # programming language, in the VisualStudio development environment.

The results obtained in the bachelor's thesis can be used as ancillary software for dormitory management.

Key words: dormitory management, automatic document generation, debt monitoring.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. На сьогоднішній день життя без житла не можна уявити. Житло є одним з основних показників благополуччя. Але не в кожної людини є своє власне житло і тому часто виникає проблема пошуку житла для тимчасового проживання. Для людей, що здають житло в оренду, це є можливістю для заробітку. Здавати одне житло в оренду не є складною задачею, але у випадку значної кількості об’єктів нерухомості проблема керування процесом здачі в оренду суттєво ускладняється. Потрібно наймати додатковий персонал для обробки документів та моніторингу оплати. Вирішити цю задачу та оптимізувати процес керування орендою житла допомагають спеціалізовані програмні додатки [1].

За статистикою, близько 70% студенів, які приїжджають навчатися в інше місто, повинні займатись пошуком тимчасового житла на період навчання. Тому найкращим вибором для студента є державний гуртожиток, що надає йому місце у кімнаті на період навчання. Керуванням гуртожитком займаються багато людей, які мають різні обов’язки. На даний момент, на жаль, більшість процесів відбувається у паперовому вигляді, що є надзвичайно неефективно порівняно з тим, якби більшість процесів були б автоматизовані та переведені у цифровий формат [2]. Програмний додаток для керування гуртожитком дозволяє зробити цей процес більш ефективним та простим, а персонал може витрачати менше часу на рутинні задачі з підтримки функціонування гуртожитку. Для самих студентів взаємодія з гуртожитком теж стає більш зручною. Цифровізація процесу керування гуртожитком робить його прозорим для усіх учасників цього процесу, що відповідає сучасним трендам в Україні.

Найбільш поширеною формою програмних додатків серед комерційних аналогів є веб-застосунки. Веб-застосунки є найкращим вибором, оскільки не потребують додаткових завантажувань для користувача при роботі з додатком. Також веб-інтерфейси є найбільш поширеними та найбільш зрозумілими для користувачів, що робить взаємодію користувача з додатком ефективнішою.

Розробка програмного забезпечення для веб-застосунків [3] є найбільш поширеною та найлегшою у підтримці серед розробників програмного забезпечення, оскільки сайт відкривається за допомогою браузера на всіх популярних платформах. А у випадку розробки програмного додатку для ПК можуть виникати додаткові труднощі при розробці додатку для ОС Windows та Linux. Також для зручності отримання інформації про новини з гуртожитку, потрібно розробити мобільний додаток за допомогою Flutter, оскільки цей фреймворк є одним з лідерів для розробки додатків на телефони.

Таким чином, розробка додатку для керування гуртожитком з функціональними можливостями для моніторингу боргів студента та автоматичної генерації документів є досить актуальною задачею.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою дослідження є підвищення ефективності процесу керування гуртожитком шляхом реалізації функціональних можливостей для моніторингу боргів студента та автоматичної генерації документів для керування гуртожитком.

Відповідно до поставленої мети в бакалаврській дипломній роботі потрібно вирішити такі завдання:

- проаналізувати підходи та додатки для керування гуртожитком;
- розробити структуру інтерфейсу програмного додатку;
- розробити метод та алгоритми моніторингу боргів студента;
- розробити алгоритм автоматичної генерації документів для керуванням гуртожитком;
- реалізувати можливість авторизації та аутентифікації користувача;
- розробити програмне забезпечення для керування гуртожитком на основі запропонованих методу та алгоритмів;
- провести тестування програмного продукту;
- розробити інструкцію користувача.

Об’єкт дослідження – процес керування гуртожитком.

Предмет дослідження – методи та засоби розробки програмного додатку для керування гуртожитком.

Методи дослідження. У процесі дослідження використовувались: теорія алгоритмів для розробки та вдосконалення алгоритмів програмного забезпечення; комп’ютерне моделювання для перевірки та аналізу теоретичних положень; методи тестування для перевірки працездатності програмного забезпечення.

Новизна отриманих результатів.

1. Уперше запропоновано метод моніторингу боргів студента по гуртожитку, у якому нарахування боргу відбувається автоматично щомісяця, а списання боргу при підтвердженні квитанції про сплату, що дозволило підвищити точність нарахування боргу та спростити процес моніторингу боргів.

2. Удосконалено алгоритм автоматичного генерування документів для керування гуртожитком, який, на відміну від інших, дозволяє завантажувати документ лише після його підпису усіма сторонами, що дало можливість підвищити надійність та достовірність автоматично згенерованих документів.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено програмні засоби для підвищення ефективності керування гуртожитком.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У наукових працях, опублікованих у співавторстві, автору належать такі результати: використання атоматичної генерації документів для підвищення ефективності менеджменту [4], порівняльна характеристика реляційних та NoSQL баз даних [5], особливості використання технології airflow для моніторигу та планування робочих процесів [6].

Апробація результатів роботи. Результати роботи доповідалися на:

– LI науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2022 р., м.Вінниця);

– XLIX Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2020 р., м.Вінниця);

– XXII Всеукраїнська науково-технічна конференція молодих вчених, аспірантів і студентів «Стан, досягнення і перспективи інформаційних систем і технологій» (2022 р., м.Одеса).

Публікації. Основні результати досліджень опубліковано в 3 (трьох) наукових працях у збірниках матеріалів конференцій.

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз стану додатків для керування орендою житла

Кожній людині потрібні комфортні умови життя. Одним з основних критеріїв комфортного життя є наявність власного житла. Бувають ситуації, коли власного житла немає і потрібно орендувати помешкання. Для орендарів немає ніякої проблеми при здачі одного житла. Але якщо людей, які орендують житло, багато і помешкань, які повинні орендуватися, багато, то виникають проблеми з керуванням цим процесом. Проблеми моніторингу заборгованості перед орендарем, обробка документів для орендаря, моніторинг стану апартаментів та інвентарю житла – це ряд задач, які програмний додаток для керування орендою житла повинен спростити.

Не дивлячись на високий попит у комерційному програмному забезпеченні для менеджменту оренди житла, у державних структурах, такому питанню, як керування гуртожитком, приділяється занадто мало уваги. Програмні засоби менеджменту оренди житла широко застосовуються великими корпораціями та звичайними користувачами. Вони наділені досить широким функціоналом моніторингу боргів, менеджменту документів, моніторингу стану помешкання та рушієм пошуку найкращих варіантів житла для користувача та іншим функціоналом.

Відсутність програмного забезпечення для керування орендою житла для державних установ є досить великою проблемою, вказує на те, що питання керування гуртожитком стоїть відкритим. Система керування гуртожитком вирішить надзвичайно багато проблем як для персоналу підтримки функціонування гуртожитку, так і для студентів. Персонал отримає можливість швидкого доступу до інформації студента, а студент буде швидко отримувати інформацію про свій борг. Надзвичайно популярним рішенням взаємодії з додатком є веб-браузер.

Веб-застосунками називають додатки, з якими користувач може взаємодіяти через спеціальне програмне забезпечення типу браузерів. Такий тип взаємодії є кращим, ніж програмний додаток та мобільний застосунок, оскільки користувач не потребує завантаження додаткового програмного забезпечення для взаємодії з додатком. Приклад інтерфейсу веб-застосунку наведено на рисунку 1.1.

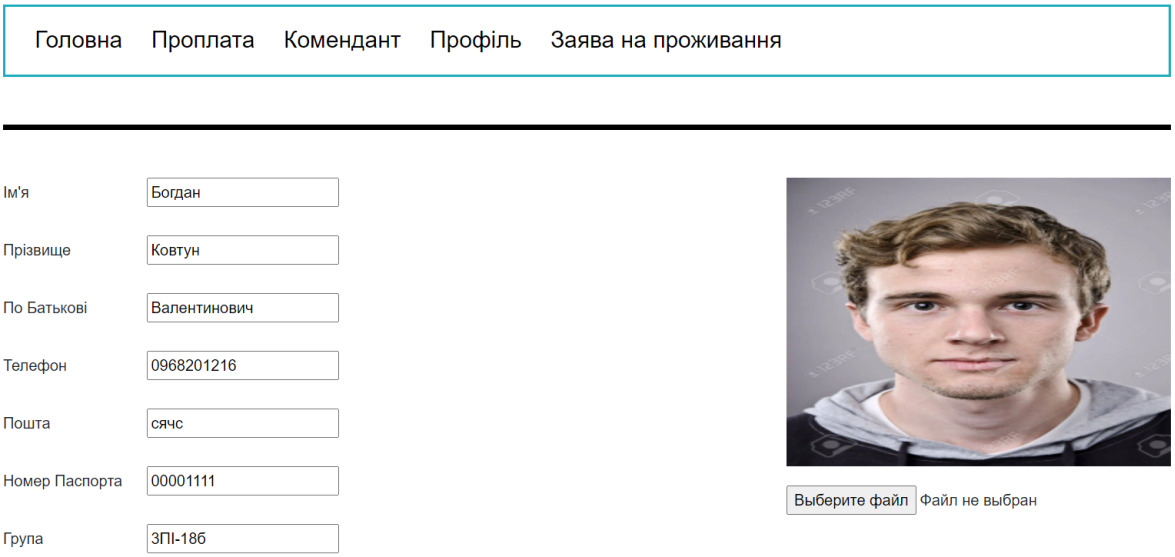


Рисунок 1.1 – Приклад інтерфейсу Веб-застосунку

Отже, проблема керування орендою житла для державних установ стоїть досить гостро і потрібно розробити систему для керування гуртожитком, що спростить життя студентам та персоналу, який обслуговує гуртожиток.

1.2 Аналіз аналогів додатку для керування гуртожитком

Найбільша кількість рішень ґрунтується на пошуку житла, а не на менеджменті оренди. Також майже не існує аналогів, що використовують державні установи. Серед існуючих рішень найбільш близькими та популярними є рішення:

– Booking.com;

– Aiosell;

– COSMO;

– CloudBeds.

Booking.com – це додаток для бронювання житла, готелів, хостелів. До переваг даного застосунку можна віднести популярність та зручність у користуванні. Можливість залишати відгуки на пропозиції та оцінки для користувачів. Має у собі сервіси реклами, що дають можливість швидко здати житло в оренду. До недоліків можна віднести перевантаженість інформації на сайті, затримка виплат та проблема з власниками нерухомості. Власник – «The Priceline Group». На рисунку 1.2 наведено користувацький інтерфейс додатку.

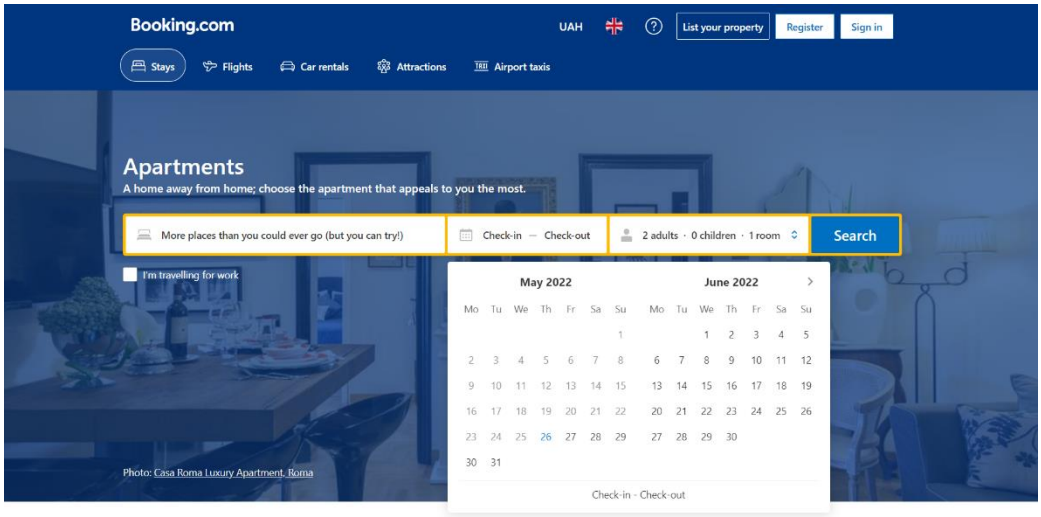


Рисунок 1.2 – Користувацький інтерфейс програми Booking.com

Aiosell – це повністю інтегрована система маркетингу готелів, яка здійснює автоматичне ціноутворення та управління доходами, щоб збільшити кількість бронювань і заповнюваність готелів, одночасно зменшуючи зусилля та неефективність. До переваг можна віднести унікальне автоматизоване ціноутворення Aiosell, розроблене власниками готелів, використовує декілька факторів для обчислення й автоматизації цін у реальному часі, як-от зайнятість, час виконання, вікно бронювання, час тижня, час бронювання, конкуренція та діапазони. Недоліками програмного продукту є ціна та відсутність моніторингу майна. На рисунку 1.3 наведено користувацький інтерфейс додатку.

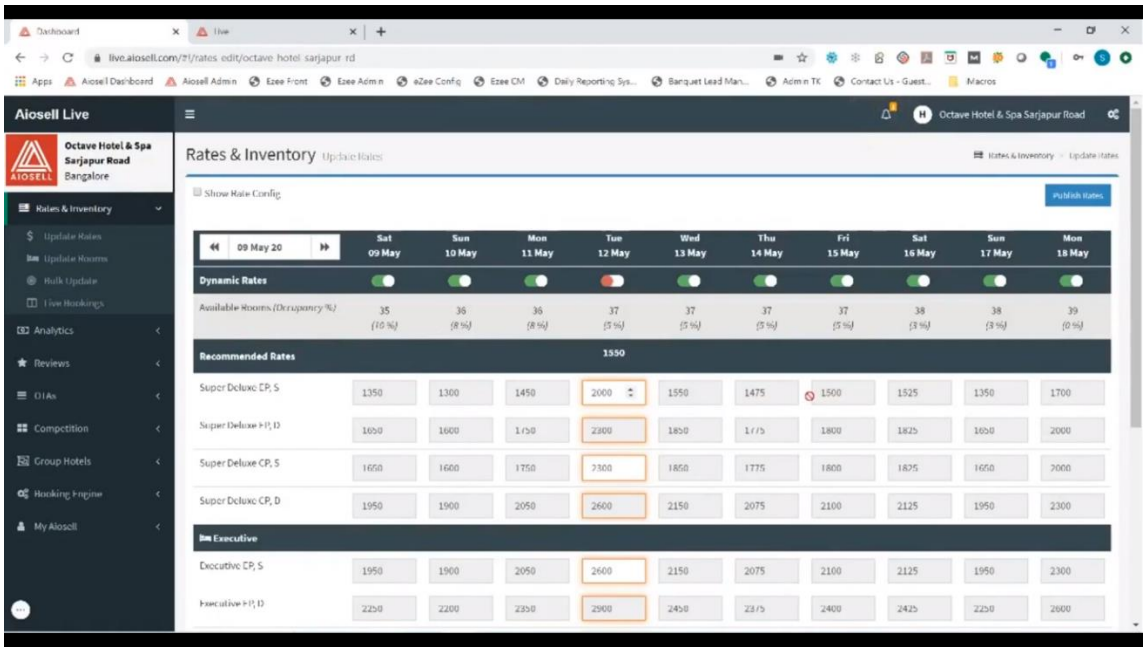


Рисунок 1.3 – Користувацький інтерфейс програми Aiosell

Cosmo – це інтуїтивно зрозуміла хмарна система управління готельним майном, розроблена для готелів усіх розмірів. Перевагами є моніторинг майна, доступність 24 години на добу, рекламна компанія для готелів. Недоліками є ціна та складність у використанні. На рисунку 1.4 наведено користувацький інтерфейс додатку.

Рисунок 1.4 – Користувацький інтерфейс програми Cosmo

Система управління нерухомістю Cloudbeds (PMS) — це хмарне рішення реєстрації майна. З його допомогою можна зареєструвати та виписати клієнтів. Використовується, щоб надсилати вільні кімнати до системи бронювання в Інтернеті та менеджера каналів. Це робить ціни та асортимент однаковими у готелі, на веб-сайті та в OTA. Перевагами є адміністрування та менеджмент фінансів, а недоліками – ціна та складність у використанні. На рисунку 1.5 наведено користувацький інтерфейс додатку.

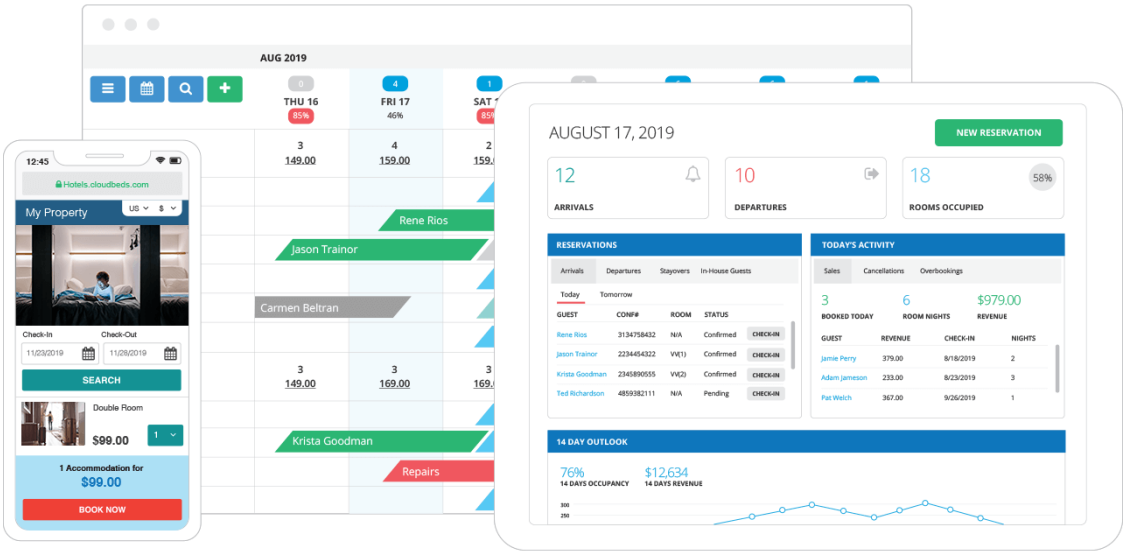


Рисунок 1.5 – Користувацький інтерфейс програми Cloudbeds

Після проведення аналізу аналогів було визначено переваги та недоліки та проведено порівняння з додатком для керування гуртожитком. Результат порівняння зведено в таблицю 1.1.

Таблиця 1.1 – Порівняльні характеристики програмних продуктів

Критерій	Booking.com	Aiosell	Cosmo	CloudBeds	Додаток для керування гуртожитком
Можливість створення профілю користувача	1	0	0	0	1

Продовження таблиці 1.1

Генерація документів	0	0	1	1	1
Автоматичне нарахування боргу	0	0	0	0	1
Можливість публікувати новини користувачам	1	1	1	1	1
Менеджмент користувачів	1	0	0	0	1
Підсумковий результат	3	1	2	2	5

Результатом аналізу та порівняння аналогів є висновок, що розробка додатку для керування гуртожитком є досить актуальною задачею. В результаті розробки отримаємо програмний продукт, який долає недоліки аналогів.

1.3 Аналіз методів розв’язання поставленої задачі

Існує кілька різних варіантів вирішення проблеми моніторингу боргу студента. Першим і найбільш очевидним є ручна зміна боргу студента, тобто комендант буде вписувати борг студента при щомісячному списанні грошей за проживання за місяць. Даний підхід є неефективним, тому що потребує від користувача зайвих кроків, коменданту потрібно буде самостійно контролювати борг студента. Без функції автоматизації процесу ведення боргу неможливо ефективно керувати гуртожитком [7].

Найбільш ефективним розв’язком задачі є створення спеціального сервісу, що в автоматичному режимі буде нараховувати борг студента та списувати борг студента при затвердженні квитанції комендантом. Для вирішення задачі було прийнято рішення розробити два сервіси: сервіс щомісячного нарахування боргу та сервіс квитанцій. Обидва сервіси взаємодіють з базою даних. Для роботи сервісу нарахування боргу потрібно в базі даних вказати ціну проживання за місяць. Сервіс буде запускатись кожного місяця, брати з бази даних ціну за

проживання за поточний місяць, для кожного проживаючого студента буде збільшувати борг у розмірі ціни за місяць. Останнім кроком буде помітка в базі даних, що сервіс виконав свою роботу за поточний місяць.

Програмний модуль, що є відповідальним за генерування документів для керування гуртожитком, було вирішено зробити таким, щоб він міг створювати документи, використовуючи поточні дані користувачів. Це дозволить суттєво пришвидшити роботу персоналу при керуванні гуртожитком.

Реалізація програмного продукту керування гуртожитком у вигляді веб-застосунку є найбільш зручним та найбільш ефективним вибором платформи. Адже веб-застосунки є найбільш популярною та простою платформою взаємодії користувача з додатком. Також платформа веб-застосунків є найбільш простою при розробці серед інших платформ, таким чином розробники зможуть швидше та ефективніше додавати функціонал до додатку.

Будь яка взаємодія користувача з додатком для керування гуртожитком буде означати оброблення HTTP запитів користувача [8]. Доступ до самого додатку можна буде отримати ввівши доменну адресу веб-застосунку. Запити та відповіді на запити будуть включати в собі JSON-файли, що будуть мати у собі всю інформацію необхідну для відображення сторінки. Також для роботи з додатком для керування гуртожитком потрібно буде мати вже створений акаунт користувача. Створювати акаунти має право лише адміністратор. Перед початком роботи з додатком потрібно буде ввести логін та пароль користувача.

Існує декілька рішень завдання перегляду новин про гуртожиток, одне з найочевидніших це додати новини як окрему сторінку у веб-додатку, але для того, щоб переглянути новину користувачу потрібно зробити багато непотрібних дій. Тому найкращим рішенням для того щоб забезпечити користувачу швидкий доступ до новин, це розробка мобільного застосунку для перегляду актуальних новин про гуртожиток. Також мобільний додаток буде розроблятися на фреймворці Flutter, оскільки це один з найпопулярніших фреймворків.

Існує велика кількість бібліотек та вбудованих програмних інструментів для відображення сторінки користувача та реалізації модуля авторизації та

аутентифікації. Різні бібліотеки мають свої переваги та недоліки. Для розробки додатку для керування гуртожитком важливими є швидкість роботи додатку, безпека додатку, простота в підтримці програмного коду додатку. Для задоволення цих потреб потрібно використовувати фронтенд фреймворк React, що є популярним для вирішення фронтенд задач та IdentityServer [9], що є чудовим рішенням для авторизації та аутентифікації користувачів, також ASP NET [10] буде використовуватися в якості бекенд фреймворку [11].

1.4 Постановка задач розробки додатку для керування гуртожитком

Після проведення аналізу питання розробки програмного забезпечення для керування гуртожитком, було визначено наступні завдання, які необхідно виконати для розробки програмного продукту:

- розробити структуру інтерфейсу програмного додатку;
- розробити метод та алгоритми моніторингу боргів студента;
- розробити алгоритм автоматичної генерації документів для керуванням гуртожитком;
- реалізувати можливість авторизації та аутентифікації користувача;
- розробити програмне забезпечення для керування гуртожитком на основі запропонованих методу та алгоритмів;
- провести тестування програмного продукту;
- розробити інструкцію користувача.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі було зроблено аналіз стану додатків для керування орендою житла. Також було проведено аналіз існуючих аналогів і їх порівняння між собою та додатком для керування гуртожитком. Було доведено доцільність розробки, а також здійснено аналіз різних підходів для вирішення завдання. Було визначено основні завдання, що потрібно зробити при розробці програмного продукту.

2 РОЗРОБКА СТРУКТУРИ, МЕТОДУ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

2.1 Розробка інтерфейсу програмного додатку

Доступ до програмного додатку буде надано за допомогою браузера. Користувач буде відкривати веб сторінку додатку і буде бачити весь функціонал. Сервер буде надсилати стилі та розмітку для побудови сторінки. Користувач може взаємодіяти з сервером завдяки кнопкам. Користувач зможе відсилати та отримувати дані від сервера завдяки запитам POST та GET. При розробці інтерфейсу було вибрано вебінтерфейс як вид інтерфейсу. Вебінтерфейс [12] – це інтерфейс, що відкривається за допомогою браузера і користувач може взаємодіяти з сайтом та веб додатком завдяки цьому. Вебінтерфейси широко застосовуються оскільки для роботи з додатком не потрібно завантажувати додаткове програмне забезпечення, а лише потрібно відкрити сторінку в браузері. При розробці інтерфейсу було розроблено анімації для показу коректності роботи додатку та для естетичного вигляду.

Для створення вебінтерфейсів застосовують HTML як каркас, за допомогою CSS можна надати сайту гарного вигляду, а завдяки JavaScript дозволи взаємодіяти з серверною частиною додатку. Кожен браузер має свої нюанси з HTML та CSS і це викликає проблеми з розробкою вебінтерфейсу, тому потрібно розробляти вебінтерфейс який буде виглядати однаково на різних браузерах. Крім того, можливість користувача змінювати багато налаштувань сторінки браузера (наприклад, розмір шрифту, кольору, відключення підтримки сценаріїв) може завадити коректному відображенню вебінтерфейсу.

Для розробки вебінтерфейсу було використано фреймворк React. Оскільки цей фреймворк дозволяє будувати динамічні веб застосунки з естетично сучасним UI та з backend функціоналом. React – це фреймворк для створення клієнтської частини додатку, що вирішує проблему оновлення вмісту вебсторінки без перезавантаження сторінки, що є досить важливою при створенні односторінкових додатків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників. Також було використано

інструментарій React для навігації по веб застосунку, зміни даних на сторінці без перевантаження сторінки, розробки форм вводу даних. React має на собі мету полегшувати процес розробки односторінкових додатків, що мають у собі динамічні дані, що змінюються в процесі взаємодії з додатком, без перезавантаження сторінки. Основна мета є в тому, щоб поєднувати швидкість, масштабованість та простоту. React [13] обробляє тільки інтерфейс користувача у додатках. Це відповідає видові у шаблоні модель-вид-контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови вебзастосунків. Також React широко використовують у поєднанні з іншими інструментами для розробки додатків, наприклад Redux.

Таким чином, вебінтрейс є одним з найпоширеніших видів інтерфейсів і також повністю відповідає вимогам для розробки програмного забезпечення для керування гуртожитком. React є найефективнішим фреймворком при розробці додатку, оскільки дозволяє створювати динамічні сторінки та оновляти дані без перезавантаження сторінки.

2.2 Розробка структури графічного інтерфейсу програмного додатку

Загальна структура сайту для керування гуртожитком буде аналогічною до структури вікна звичайного сайту. Структура складається із трьох частин: хедера, робочої області та футера.

У хедері сайту буде зображено лого та кнопки для регістрації та входу в додаток. Також після в додаток входу з’явиться навігаційний список. Для різних користувачів навігаційний список буде різний. Також при нажатті на кнопку «Вхід» відкриється сторінка з полями для вводу логіну та паролю. Токож у футері знаходиться інформація про розробників. Графічна схема головного вікна сайту наведена на рисунку 2.1.

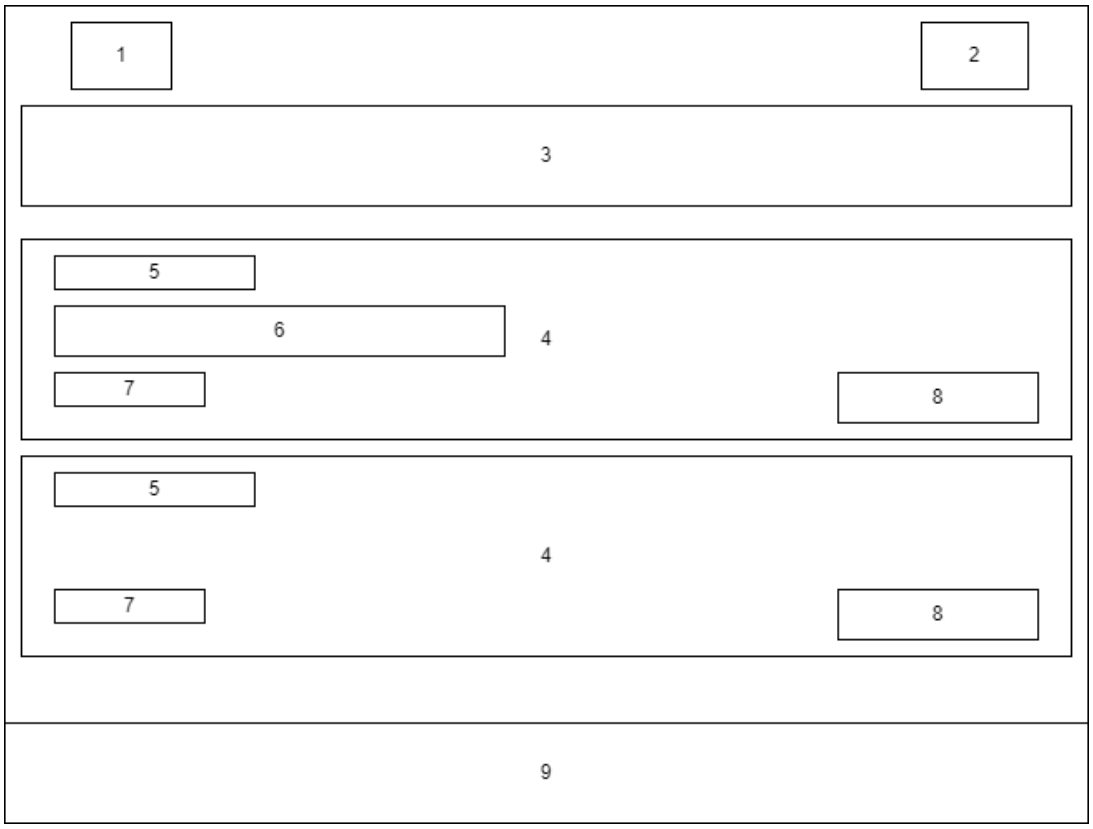


Рисунок 2.1 – Графічна схема головного вікна сайту

Основні елементи інтерфейсу головної сторінки:

1. Лого.
2. Кнопки керування входу та виходу.
3. Меню.
4. Карта з повідомненням
5. Заголовок повідомлення.
6. Текст повідомлення.
7. Час повідомлення.
8. Автор повідомлення.
9. Інформація про авторів сайту.

Для того, щоб користувачі могли зберігати інформацію про себе потрібно розробити сторінку з полями, які можна заповнити та натиснувши кнопку «Зберегти» зберегти інформацію про себе. Графічна схема сторінки профілю студента наведена на рисунку 2.2.

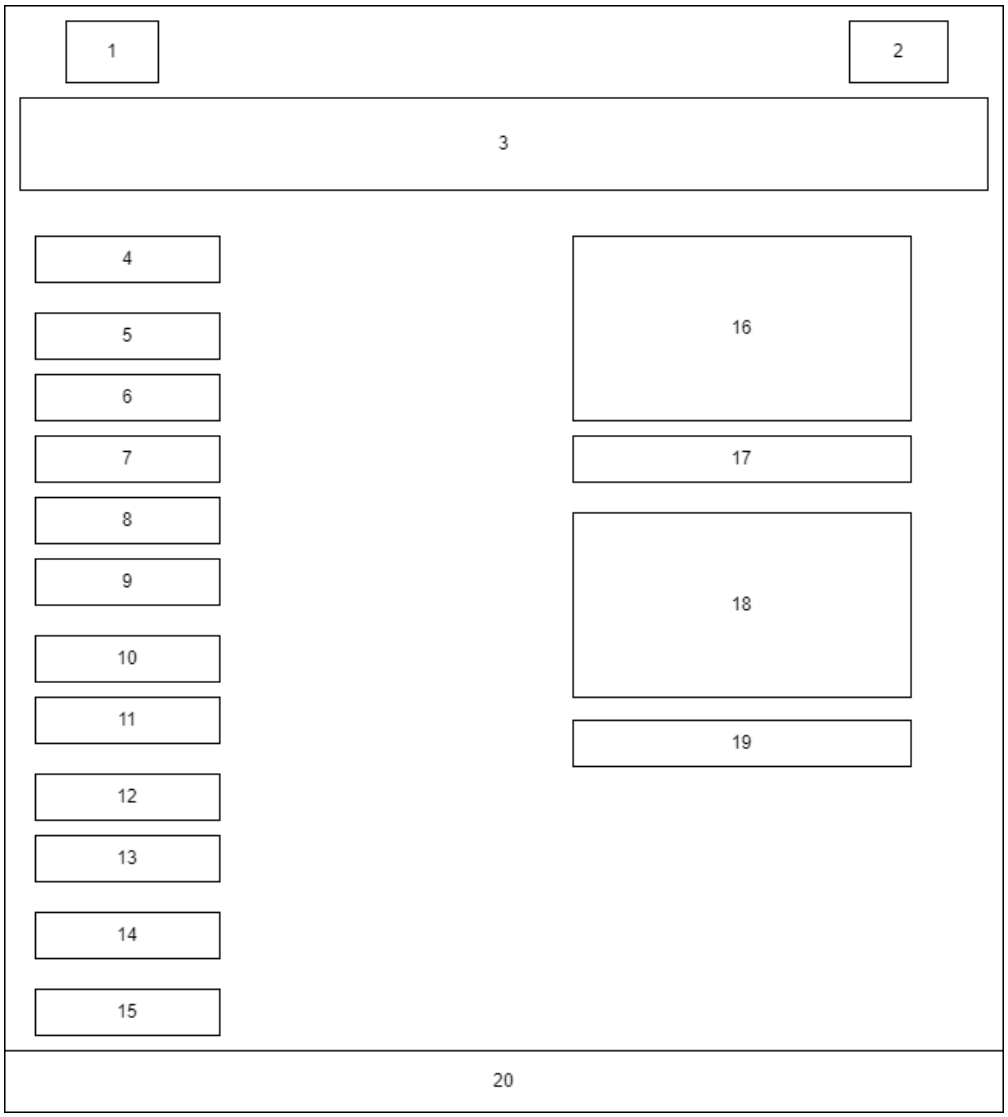


Рисунок 2.2 – Графічна схема сторінки профілю

Основні елементи інтерфейсу сторінки профілю:

1. Лого.
2. Кнопки керування входу та виходу.
3. Меню.
4. Поле вводу ім’я.
5. Поле вводу прізвища.
6. Поле вводу по батькові.
7. Поле вводу телефону.
8. Поле вводу пошти.
9. Поле вводу номеру паспорту.

10. Поле вводу групи
11. Поле вводу факультету.
12. Поле вводу Дати отримання паспорту.
13. Поле вводу дати народження.
14. Поле вводу курсу.
15. Кнопка «Зберегти».
16. Фотографія студента.
17. Кнопка збереження фотографії студента.
18. Фотографія підпису студента.
19. Кнопка збереження фотографії підпису.
20. Інформація про авторів сайту.

Також для того, щоб студент міг завантажувати квитанції коменданту гуртожитку потрібно створити окрему сторінку. Це допоможе автоматизувати процес списання боргу та підвищити процес моніторингу оплати боргу студентом. Графічна схема сторінки завантаження квитанцій представлена на рисунку 2.3.

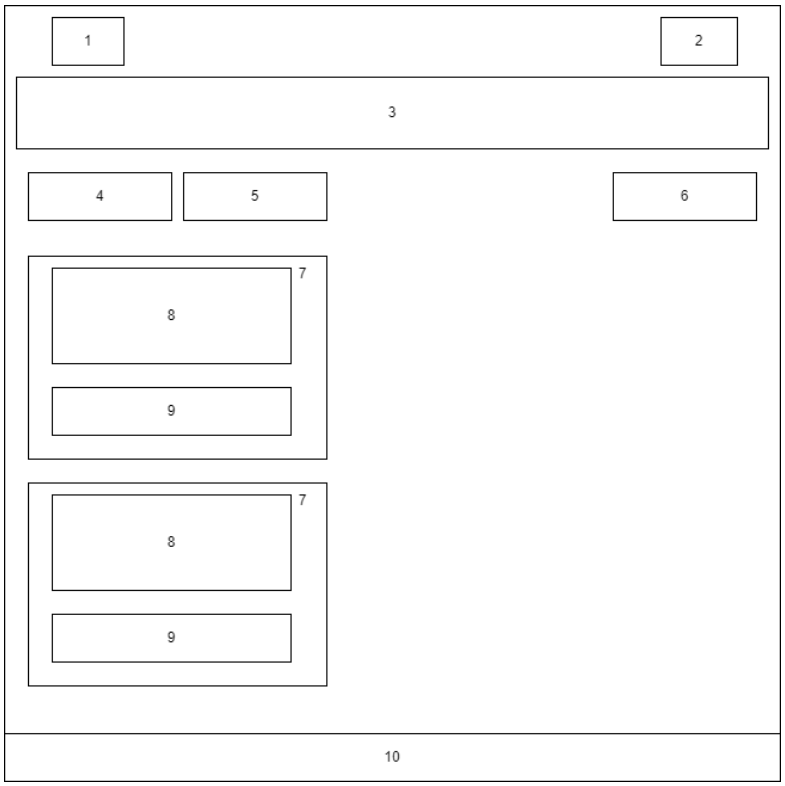


Рисунок 2.3 – Графічна схема сторінки завантаження квитанцій

Основні елементи інтерфейсу сторінки завантаження квитанцій:

1. Лого.
2. Кнопки керування входу та виходу.
3. Меню.
4. Поле завантаження квитанції.
5. Кнопка «Зберегти».
6. Борг студента.
7. Картка інформації про квитанцію.
8. Картинка квитанції
9. Дата завантаження квитанції.
10. Інформація про авторів сайту.

У додатку буде багато користувачів і коменданту потрібно буде займатись їх менеджментом. Найкращим вибором стане адмін панель, що буде мати у собі всю навігаційну інформацію про студентів. Таким чином керування гуртожитку стане набагато простішим. Графічна схема сторінки адмін панелі представлена на рисунку 2.4.

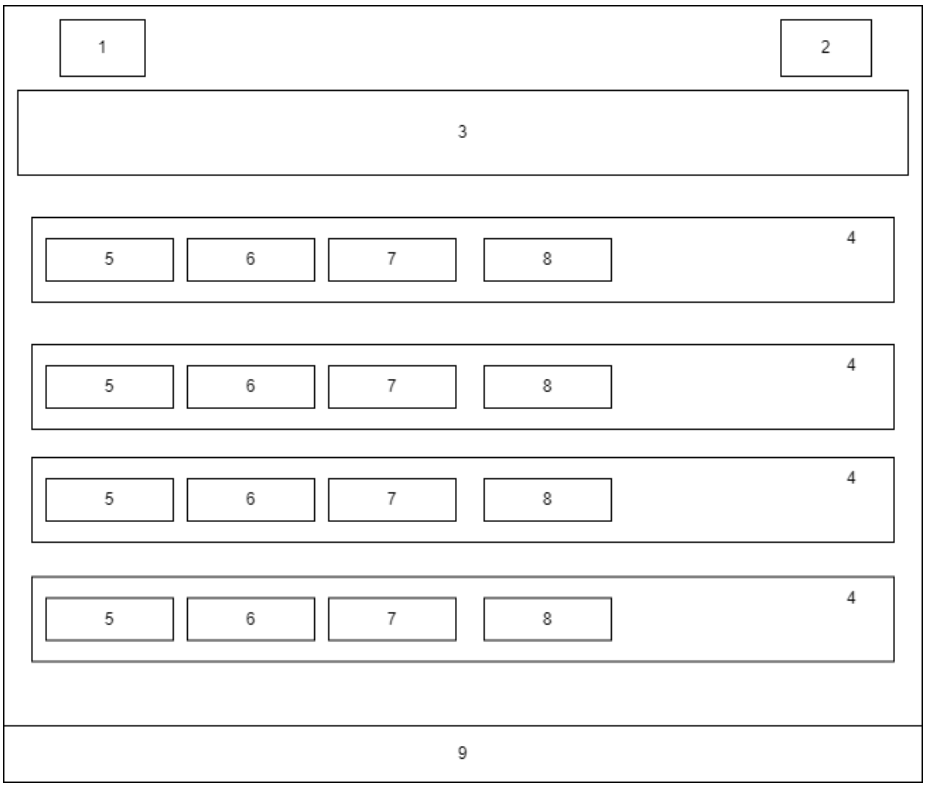


Рисунок 2.4 – Графічна схема сторінки адмін панелі

Основні елементи інтерфейсу сторінки адмін панелі:

1. Лого.
2. Кнопки керування входу та виходу.
3. Меню.
4. Карта навігації студента.
5. Назва студента.
6. Кнопка «Профіль».
7. Кнопка «Проплата».
8. Кнопка «Заява на поселення».
9. Інформація про авторів стайту.

Також для швидкого отримання інформації про гуртожиток для мобільних пристроїв був розроблений інтерфейс для Flutter. Інтерфейс був розроблений максимально зрозумілим та зручним, щоб користувачі могли отримувати актуальні новини про гуртожиток і це було б зручно. Графічна схема сторінки нотифікацій для Flutter представлена на рисунку 2.5.

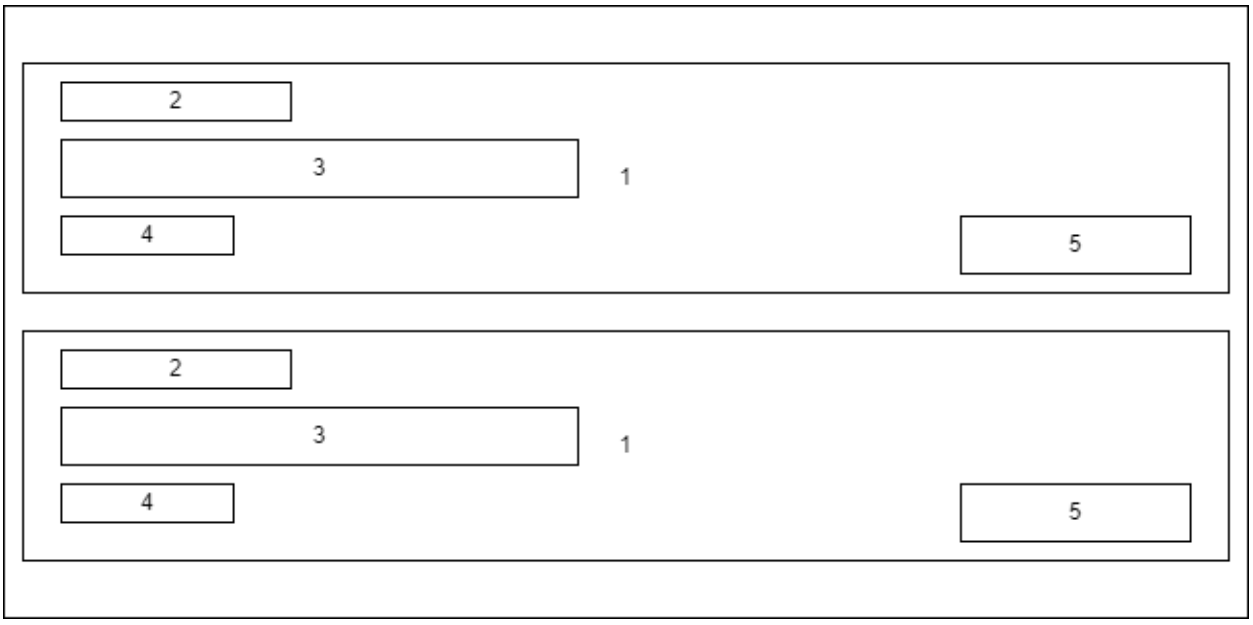


Рисунок 2.5 – Графічна схема сторінки нотифікацій для Flutter

Основні елементи інтерфейсу сторінки нотифікацій для Flutter:

1. Карта з повідомленням.

2. Заголовок повідомлення.

3. Текст повідомлення.

4. Час повідомлення.

5. Автор повідомлення.

Одним з найважливіших етапів при розробці додатку для керування гуртожитком є розробка інтерфейсу, тому що інтерфейс це один з ключових елементів програмного продукту, що відповідає за взаємодію користувача з продуктом. Тому дуже важливо зробити інтерфейс максимально зручним та зрозумілим, оскільки це зробить керування гуртожитку більш ефективнішим і покращить життя студентам.

2.3 Розробка алгоритму створення документів для керування гуртожитком

Одна з найважливіших функціональностей програмного додатку для керування гуртожитком є алгоритм створення документів. Алгоритм створення документів представлений на рисунку 2.6.

Алгоритм створення документів складається з наступних кроків:

Крок 1. Початок.

Крок 2. Отримати від студента інформацію про особу прізвище, ім’я, по батькові, паспортні дані і фотографію підпису.

Крок 3. Перевірити правильність введених даних. Якщо дані коректні – перейти до кроку 4, якщо ні – до кроку 14.

Крок 4. Додати в базу даних інформацію про студента та фотографію підпису.

Крок 5. Отримати від коменданта інформацію про особу прізвище, ім’я, по батькові та фотографію підпису.

Крок 6. Перевірити правильність введених даних. Якщо дані коректні – перейти до кроку 7, якщо ні – до кроку 14.

Крок 7. Додати в базу даних інформацію про коменданта та фотографію підпису.

Крок 8. Витягнути з бази даних інформацію про студента та коменданта включаючи фотографії підписів.

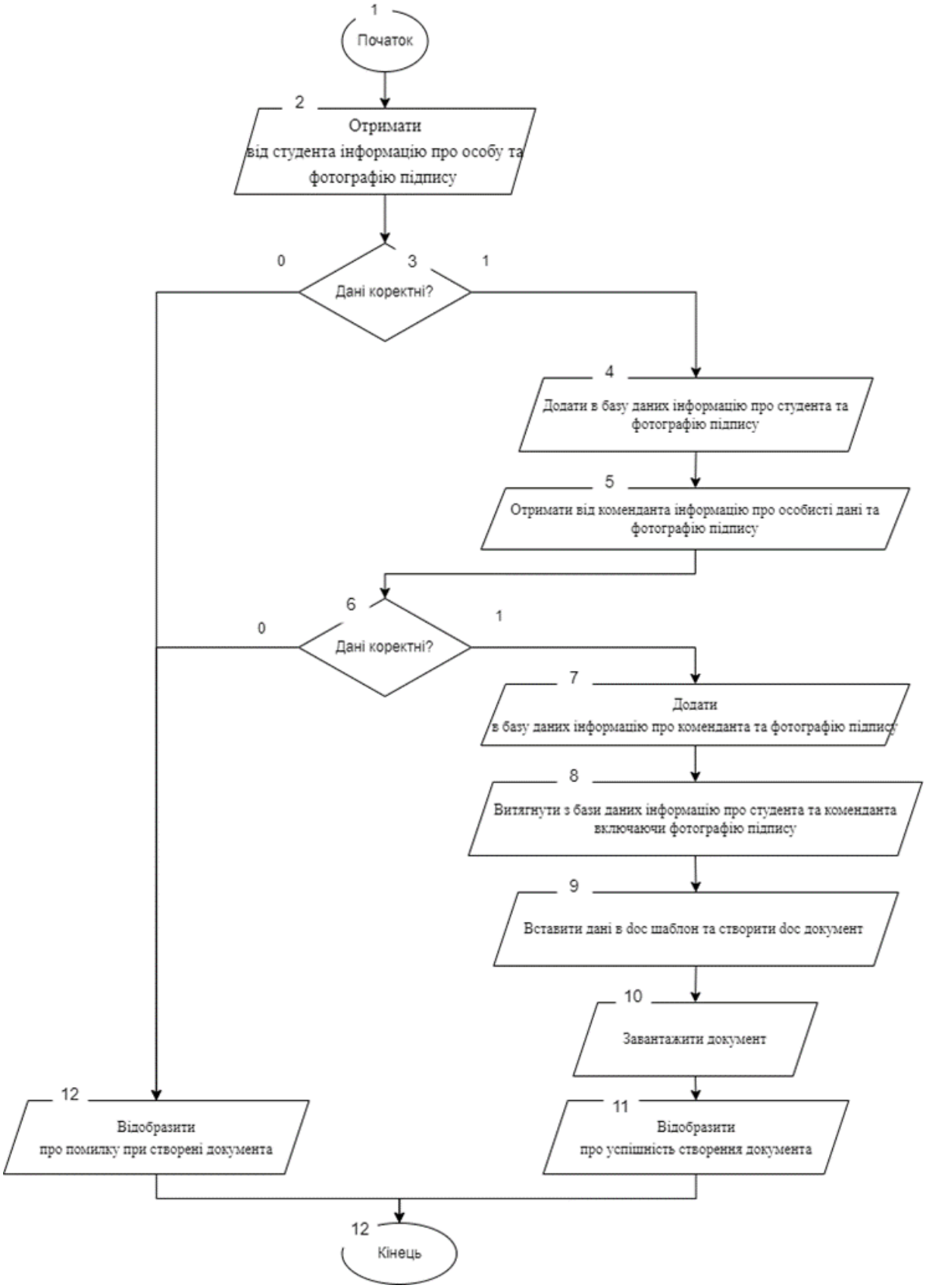


Рисунок 2.6 – Блок-схема алгоритму створення документів

Крок 9. Вставити дані в doc шаблон та створити doc документ.

Крок 10. Завантажити документ.

Крок 11. Відобразити про успішність створення документа.

Крок 12. Відобразити про помилку при створені документа.

Крок 13. Кінець.

2.4 Розробка методу та алгоритмів моніторингу боргів студента

Однією з найважливіших функціональностей додатку керування гуртожитком є функціональність моніторингу боргів студентів. Функціональність моніторингу боргів включає в себе систему знаття з балансу студента коштів за місяць проживання в гуртожитку. Списання коштів за місяць відбувається кожного місяця та відбувається лише один раз. Також включає в себе систему нарахування коштів шляхом додавання в базу квитанції про оплату.

Для програмної реалізації функції зняття коштів з балансу студента було розроблено алгоритм, що списує з балансу студента кошти за поточний місяць, представлено на рисунку 2.7. Виконання даного алгоритму виконується кожного місяця о 1 годині ранку. Планування виконання даного алгоритму буде виконуватися за допомогою хмарного рішення Airflow [14].

Алгоритм зняття коштів з балансу студента складається з наступних кроків:

Крок 1. Початок

Крок 2. Комендант встановлює ціну за проживання на поточний місяць.

Крок 3. Вибір з бази активованих студентів

Крок 4. Запустити на виконання цикл, який перебирає всіх студентів. Якщо досягнуто кінця списку – перейти до кроку 9, якщо ні – до кроку 5.

Крок 5. Перевірити, чи є студент активним і останній перевірений місяць не є поточним . Якщо так – перейти до кроку 6, якщо ні – до кроку 4

Крок 6. Встановити для студента останні перевірений місяць поточний місяць.

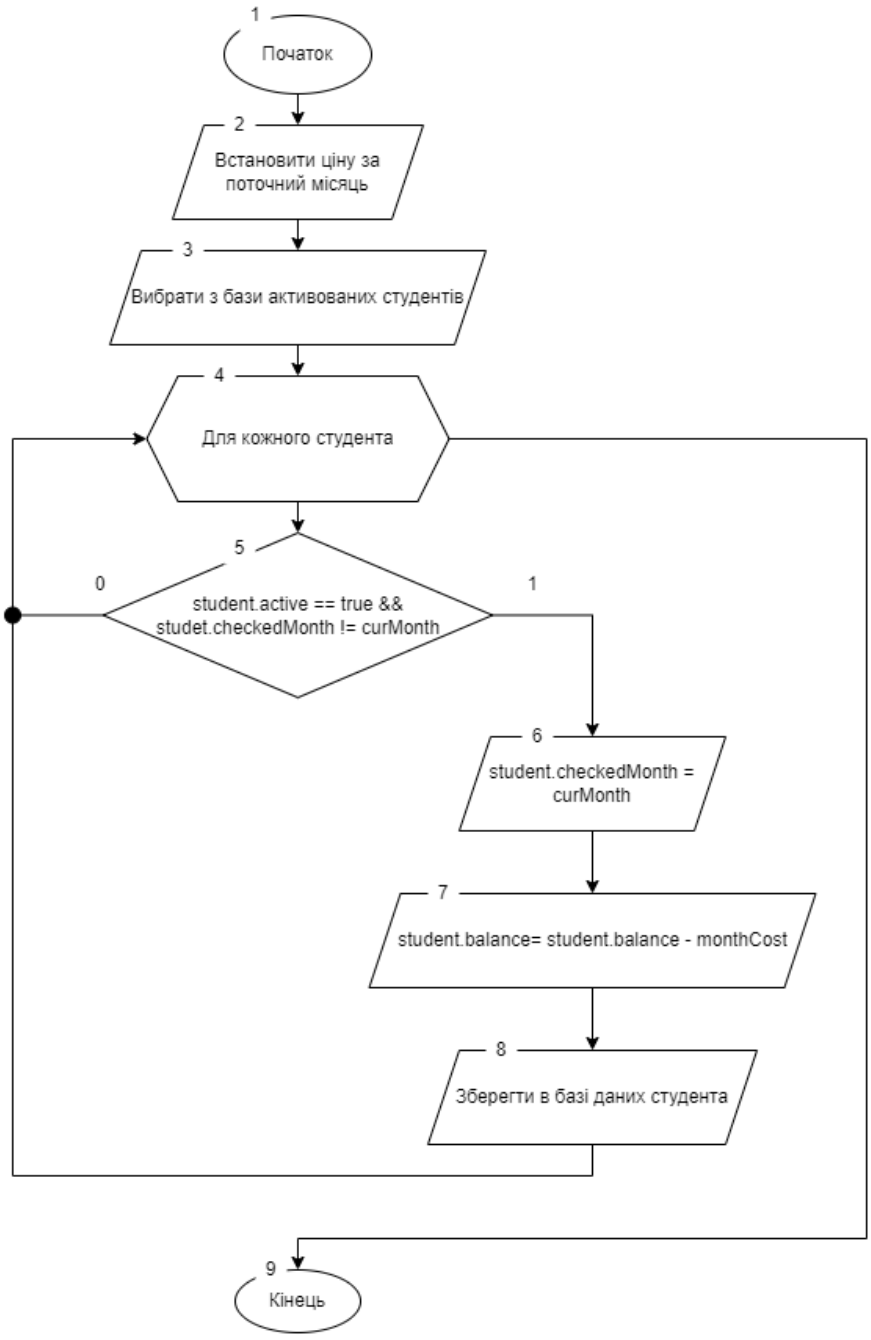


Рисунок 2.7 – Блок-схема алгоритму зняття коштів з балансу студента

Крок 7. Зменшити баланс студента на вартість проживання за поточний місяць у гуртожитку.

Крок 8. Зберегти в базі даних зміни для студента

Крок 9. Кінець.

Для програмної реалізації алгоритму, що нараховує кошти на баланс студента та додає квитанцію про оплату, представлено на рисунку 2.8. Також

присутній функціонал який надає можливість коменданту переглянути та підтвердити квитанцію і тим самим зарахувати гроші студенту.

Алгоритм нарахування коштів на баланс студента та на додавання квитанції про оплату.

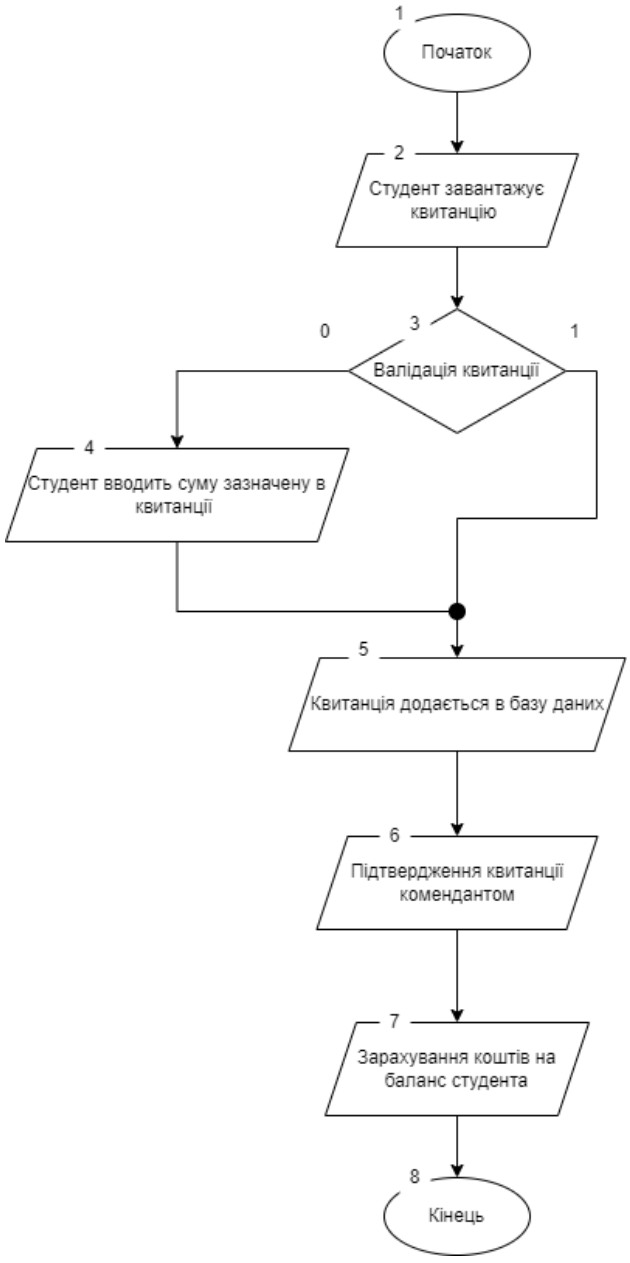


Рисунок 2.8 – Блок-схема алгоритму нарахування коштів студенту та додавання квитанції в базу даних

Крок 1. Початок.

Крок 2. Завантаження студентом квитанції.

Крок 3. Валідація квитанції. Визначення суми зазначеної в квитанції. Якщо так – перейти до кроку 5, якщо ні – до кроку 4.

Крок 4. Ввести суму зазначену в квитанції.

Крок 5. Зберегти квитанцію в базі даних

Крок 6. Підтвердження квитанції комендантом.

Крок 7. Зарахування коштів на баланс студента

Крок 8. Кінець.

2.5 Висновки

У другому розділі було проаналізовано можливі сценарії взаємодії користувача з додатком для керування гуртожитком. Було проаналізовано стандартну структуру веб-інтерфейсу, на основі якого був розроблений інтерфейс для додатку, що є зручним та зрозумілим для користувача. Був розроблений алгоритм для генерації документів для керування гуртожитком. Також було розроблено метод та алгоритми автоматизації моніторингу боргів студента, що включає у себе функціонал нарахування боргу кожного місяця та списання боргу шляхом відправки студентом квитанції про оплату.

3 РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТ

3.1 Варіантний аналіз і обґрунтування вибору мови програмування

Мова програмування є одним з найважливіших інструментів при розробці програмного забезпечення. Мова програмування повинна бути статичною, оптимізованою та об’єктно орієнтованою, оскільки на об’єктно орієнтованих мовах простіше розробляти масштабні проекти. Для розробки додатку керування гуртожитком важливим критерієм є наявність документації та навчальних матеріалів, статична типізованість, об’єктно-орієнтованість.

Розберемо такі мови програмування як C++, Java, C# , Python.

C++ [15] – мова програмування, що компілюється в машинний код та є статично типізованою, має у собі функціонал для роботи з низьким та високим рівнем програмування. Компілювання відбувається в машинний код, що робить цю мову швидкою у виконанні у порівнянні з іншими мовами. C ++ повністю сумісний із C. C ++ широко використовується для розробки додатків, операційних систем, драйверів пристроїв, відеоігор тощо.

C# [16] – мова програмування, що є об’єктно-орієнтованою. Спеціально розроблена Microsoft для .NET Framework. C# є представником мови C-подібним синтаксисом. Мова має статичну типізацію, узагальнені типи і методи, анонімні функції, підтримує перевантаження операторів, вказівники на члени функції класів.

Kotlin [17] – мова програмування, що є статично типізованою, працює на віртуальній машині JVM і розробляється компанією JetBrains. Також компілюється в JavaScript. Мова була названа на честь острова Котлін у Фінській Затоці, де розміщена частина Кронштадту. Розробники розробляли Kotlin для того щоб створити мову програмування більш лаконічнішу та типо-безпечнішу мову, ніж Java, і простішу, ніж Scala.

Python [18] – мова програмування для високого рівня, основне завдання якої це створення додатків: програм для ПК, ігор, веб-додатків, утиліт для роботи з базами даних тощо. Python широко застосовується в області машинного

навчання. Також мова Python стає все більш популярною оскільки ця мова проста та ефективна. Ця мова є динамічно типізованою, але з основних мінусів мови є її швидкість. Але швидкість виконання можна пришвидшити використовуючи різні бібліотеки.

Після аналізу мов програмування, було складено порівняльну таблицю мов програмування у таблиці 3.1, яка показує відмінності Kotlin, Python, C#, C++.

Таблиця 3.1 – Характеристики мов програмування

	Kotlin	Python	C#	C++
Можливість керування пам’яттю напряму	0	0	0	1
Можливість прямого використання бібліотек Windows.	0	0	1	1
Об’єктно-орієнтована	1	1	1	1
Статична типізація	1	0	1	1
Неявне приведення типів без втрати даних	1	1	1	1
Підтримують процедури	1	1	1	1
Популярність	1	1	1	0
Кросплатформеність	1	1	1	0
Сумарний коефіцієнт	6	5	8	6

Проаналізувавши дані таблиці, найкращою мовою програмування є C#.

Отже, після аналізу мов програмування: Kotlin, Python, C# та C++, було прийнято рішення, що C# найкраще підходить для розробки програмного забезпечення для керування гуртожитком. Також відповідає критеріям для розробки, що необхідні для успішної підтримки проекту та для швидкої розробки додатку.

3.2 Вибір середовища розробки та СКБД

Одним з найважливіших інструментів при розробці програмного забезпечення є середовище розробки або IDE. Середовище розробки – це програмний додаток для пришвидшення розробки програмного забезпечення. Складається з редактора коду, інструментів для рефекторингу, компіляції та дебагу програм. Майже всі сучасні середовища розробки мають можливість автодоповнення коду. Інтегровані середовища розробки розроблені з метою збільшення ефективності роботи програміста, надавши йому інструменти, що автоматизують процес розробки програми. Розглянемо найпопулярніші середовища розробки на мові програмування C# такі як Visual Studio, Visual Studio Code, Eclipse aCute.

Visual Studio [19] – це серія продуктів від корпорації Майкрософт, що містять у собі інтегроване середовище розробки програмного забезпечення та низку інших інструментальних засобів для підвищення ефективності розробки. Ці продукти дають змогу розробляти як консольні програми, так і програми з графічним інтерфейсом, включно з підтримкою технології Windows Forms, а також вебслужби, вебзастосунки, вебсайти та інше. Приклад інтерфейсу наведено на рисунку 3.1.

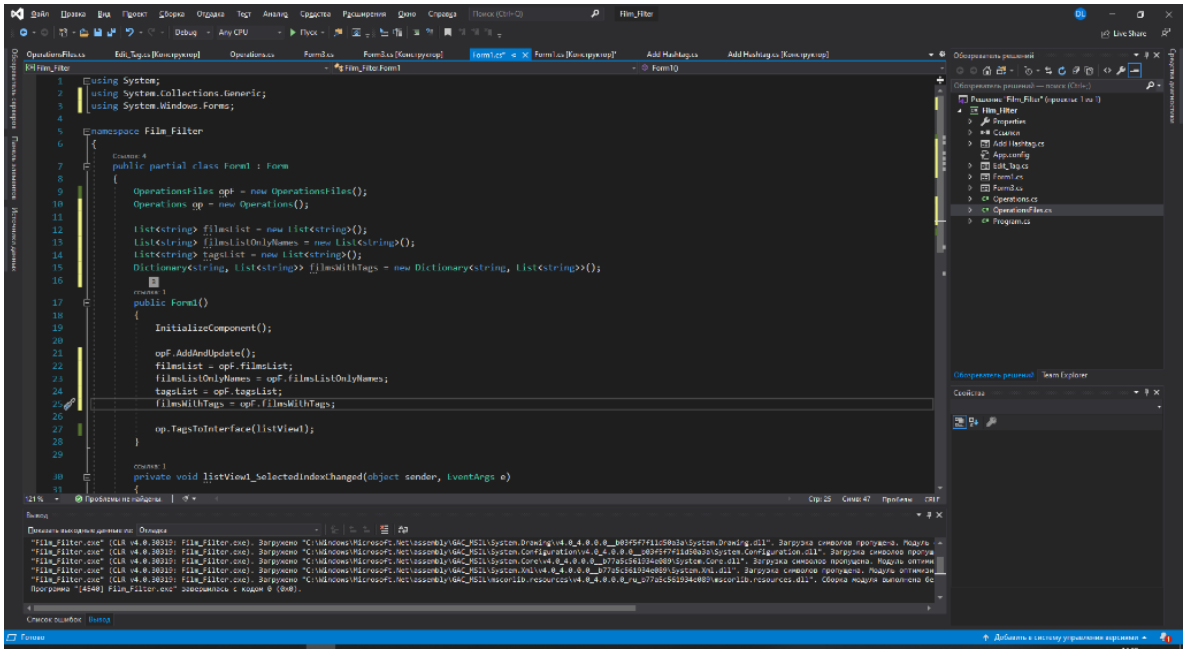


Рисунок 3.1 – Приклад інтерфейсу Visual Studio

Visual Studio Code [20] – це засіб для розробки, редагування та відлагодження сучасних вебзастосунків і програм для хмарних систем. Visual Studio Code є безкоштовним і доступним у версіях для операційних систем Windows, Linux. Компанія Microsoft презентувала Visual Studio Code у квітні 2015 на конференції Build 2015. Це середовище розробки є першим кросплатформеним продуктом у лінійці Visual Studio. Приклад інтерфейсу наведено на рисунку 3.2.

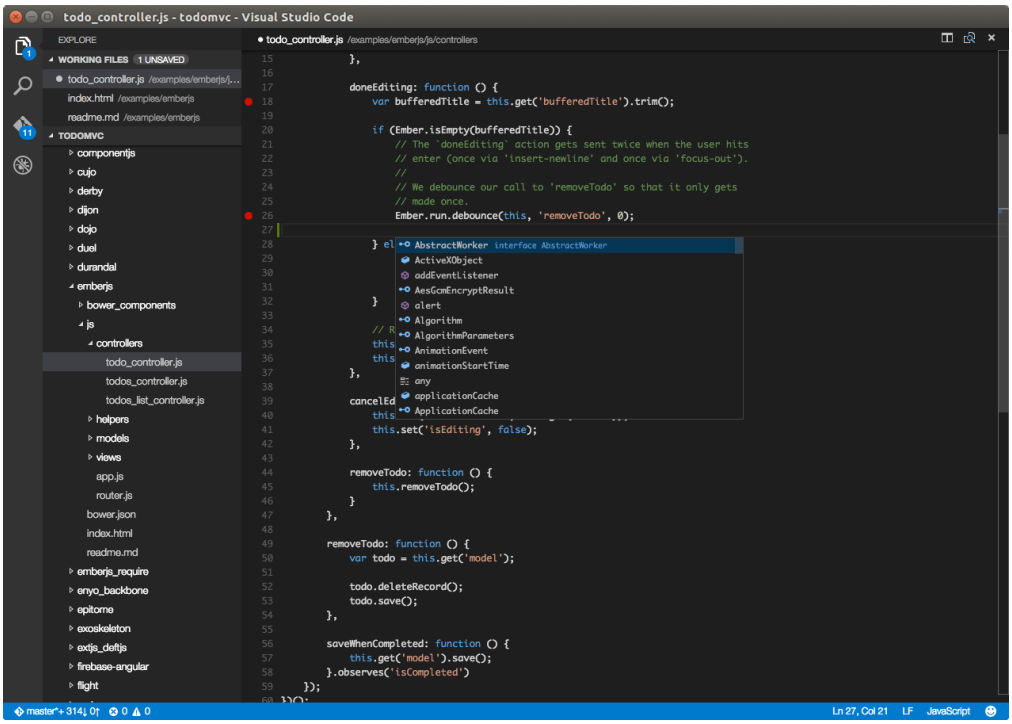


Рисунок 3.2 – Приклад інтерфейсу Visual Studio Code

Eclipse aCute [21] – це безкоштовне модульне інтегроване середовище для розробки програмного забезпечення. Розробниками є Eclipse Foundation і включає проекти, такі як платформа Eclipse, набір інструментів для програмістів на мові Java, системи контролю версій, конструктори GUI тощо. Написаний в більшості на Java, може використовуватись при розробці додатків на мові Java і, за допомогою різних плагінів, на інших мовах програмування, включаючи Python , Ada, C, C++, Ruby, C#, Fortran, COBOL, Perl, PHP, R. Приклад інтерфейсу наведено на рисунку 3.3.

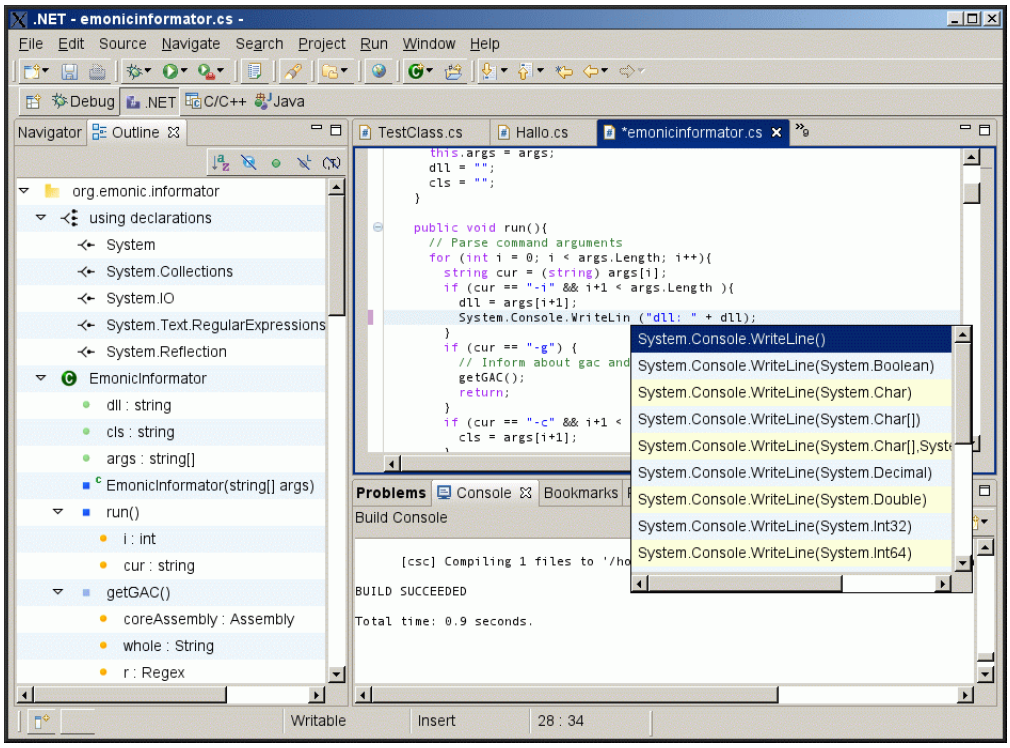


Рисунок 3.3 – Приклад інтерфейсу Eclipse aCute

Результати порівняння розглянутих інтегрованих середовищ розробки за обраними критеріями наведено в таблиці 3.2.

Таблиця 3.2 – Порівняння інтегрованих середовищ розробки

Критерій	Visual Studio	Visual Studio Code	Eclipse aCute
Безкоштовність	1	1	1
Універсальність	1	0,5	0
Кросплатформність	1	1	1
Швидкість роботи	0,5	0,5	0,5
Невимогливість до продуктивності системи	0	0	0
Магазин розширень	1	1	0,5
Автодоповнення коду	1	1	0,5
Підсумковий результат	5,5	5	3,5

За результатами наведеними в таблиці 3.2 найкращим вибором IDE є Visual Studio, оскільки це середовище є швидким та універсальним і досить добре підходить для розробки програмного забезпечення для керуванням гуртожитку.

Для розробки програмних додатків широко використовуються системи управління базами даних для ефективного менеджменту великої кількості даних. СКБД – це набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних. Надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних. Розглянемо найпопулярніші із них, а саме MSSQL, MySQL та MongoDB.

MSSQL [22] – це база даних, що розробляється компанією Microsoft. Виконує головну функцію по збереженню та наданню даних при обробці запитів від інших застосунків, які можуть підключатись да бази даних через інтернет . Для взаємодії з базою даних використовується мова запитів Transact-SQL, розроблена компаніями Sybase та Microsoft. Transact-SQL є надбудовою мови запитів SQL та реалізує стандарт ANSI / ISO. Широко використовується у малих та середніх за розмірами проектах, так і для великих проектів для зберігання даних. Багато років є конкурентоспроможним гравцем серед систем управління базами даних.

MySQL [23] – це безкоштовна база даних, розроблена компанією «ТсХ» для покращення швидкості обробки запитів. Також MySQL є альтернативою комерційним рішенням. Має присутність відкритого коду, що покращує підтримку проекту. MySQL на початку свого існування була схожа на mSQL, проте з часом вона все більше розширювалася і зараз MySQL — одна з найпоширеніших систем керування базами даних. Використовується широко для створення динамічних вебсторінок, оскільки має чудову підтримку різними мовами програмування.

MongoDB [24] – це база даних, що орієнтована на роботу з документами також ця база даних має відкритий код. У MongoDB не описуються таблиці, а додаються записи у вигляді JSON. MongoDB завоювала ринок для швидких і масштабованих систем, що мають у собі концепцію ключ та значення. Також

MongoDB має власну форму запитів, що не є гіршою за мову запитів реляційних баз даних.

Результати порівняння розглянутих систем керування базами даних за обраними критеріями наведено в таблиці 3.3.

Таблиця 3.3 – Порівняння систем керування базами даних

Критерій	MSSQL	MySQL	MongoDB
Потужність та функціонал	1	0,5	1
Швидкість читання даних	1	1	1
Легкість використання	0,5	0,5	1
Можливість використання в малих проектах	0,5	0,5	1
Підсумковий результат	3	2,5	4

За результатами у таблиці 3.3 найкращим вибором СКБД є MongoDB, оскільки поєднує у собі простоту використання та функціонал також повністю відповідає вимогам програмного додатку керування гуртожитком.

Отже, було вибрано мову програмування C# та СКБД MongoDB.

3.3 Програмна реалізація додатку для керування гуртожитком

Для реалізації програмного функціоналу додатку для керуванням гуртожитку було розроблено багато алгоритмів. Найважливішими є:

- алгоритм авторизації та реєстрації користувача
- алгоритм моніторингу боргів студента
- алгоритм генерації документів
- алгоритм створення профілю

Першим ділом фреймворк ініціалізує всі змінні та забезпечує dependency injection. Налаштування наведено на рисунку 3.4.

```
        public class Startup
            {
                public void ConfigureServices(IServiceCollection services)
                {
                    services.AddCors(options =>
                    {
                        options.AddPolicy("CorsPolicy",
                            builder => builder
                                .AllowAnyMethod()
                                .AllowCredentials()
                                .SetIsOriginAllowed((host) => true)
                                .AllowAnyHeader());
                    });
                    services.Configure<FormOptions>(o =>  // currently all set to max,
configure it to your needs!
                    {
                        o.ValueLengthLimit = int.MaxValue;
                        o.MultipartBodyLengthLimit = long.MaxValue; // <-- !!! long.MaxValue
                        o.MultipartBoundaryLengthLimit = int.MaxValue;
                        o.MultipartHeadersCountLimit = int.MaxValue;
                        o.MultipartHeadersLengthLimit = int.MaxValue;
                    });
                    services.AddControllers();

                    services.AddAuthentication("Bearer")
                        .AddJwtBearer("Bearer", options =>
                        {
                            options.Audience = "api1";
                            options.Authority = "https://localhost:5000";
                        });

                    services.AddScoped<IStudentProfileManager, StudentProfileManager>();
                    services.AddScoped<IStudentProfileService, StudentProfileService>();
                    services.AddScoped<INotificationManager, NotificationManager>();
                    services.AddScoped<INotificationService, NotificationService>();
                    services.AddScoped<IDebtManager, DebtManager>();
                    services.AddScoped<IDebtService, DebtService>();
                    services.AddScoped<IPaymentManager, PaymentManager>();
                    services.AddScoped<IPaymentService, PaymentServise>();
                    services.AddScoped<IRoomManager, RoomManager>();
                    services.AddScoped<IRoomService, RoomService>();
                    services.AddScoped<IApplicationForSettlementManager,
ApplicationForSettlementManager>();
                    services.AddScoped<IApplicationForSettlementService,
ApplicationForSettlementServiceImpl>();
                    services.AddScoped<IWordGeneratorService, WordGeneratorServiceImpl>();
                    services.AddScoped<IResidenceAgreementManager,
ResidenceAgreementManager>();
                    services.AddScoped<IResidenceAgreementService,
ResidenceAgreementServiceImpl>();
                }
```

Рисунок 3.4 – Лістинг налаштування StudentAPI

Також однією з найбільш важливою функціональністю є забезпечення авторизації та реєстрації користувача. Функціональність авторизації та реєстрації розроблялась за принципом MVC. Тобто користувач взаємодіє з UI фреймворком React, шо був написаний за допомогою JS, відправляє та отримує

моделі з контролера, що є іншим сервісом. Функція створення нового користувача представлена на рисунку 3.5.

```
        private async Task CreateUser(RegisterUserRequest registerUserRequest)
                {
                    var identityUser = new ExtendedIdentityUser()
                    {
                        Id = new ObjectId(DateTime.Now, 1, 1, 1).ToString()
                    };
                    identityUser.UserName = registerUserRequest.UserName;
                    var claims = new List<Claim>
                            {
                                new Claim(JwtClaimTypes.Role, registerUserRequest.Role)
                            };
                    identityUser.Role = registerUserRequest.Role;
                    var result = await userManager.CreateAsync(identityUser,
registerUserRequest.Password);
                    result = await userManager.AddClaimsAsync(identityUser, claims);
                }
```

Рисунок 3.5 – Функція створення нового користувача

Також фрагмент коду реєстрації для фронтенду представлений на рисунку 3.6.

```
public render() {
    return (
      <div>
        <div className='text-component'>
          <p>UserName</p>
          <input type="text" value={this.state.userName}
onChange={this.handleChangeUserName} />
        </div>
        <div className='text-component'>
          <p>Password</p>
          <input type="text" value={this.state.password}
onChange={this.handleChangePassword} />
        </div>
        <div className='text-component'>
          <p>Role</p>
          <input type="text" value={this.state.role}
onChange={this.handleChangeRole} />
        </div>
        <button onClick={this.register}>Register</button>
      </div>
    )
  }
```

Рисунок 3.6 – Функція рендеренгу створення нового користувача

Для ведення моніторингу боргів студента потрібно реалізувати можливість завантаження платежу. Контролер завантаження платежу наведений на рисунку 3.7.

```
[Authorize(Roles = "student")]
[Route("student/payment")]
[HttpPost]
public string AddPayment()
{
string userId = base.GetUserId();
byte[] paymentImage = base.GetImageFromRequestBytes();
PaymentDto paymentDto = new PaymentDto();
paymentDto.Date = DateTime.Now.ToString();
paymentDto.PaymentImage = paymentImage;
paymentDto.UserId = userId;
paymentService.AddPayment(paymentDto, userId);
return "ok";
      }
```

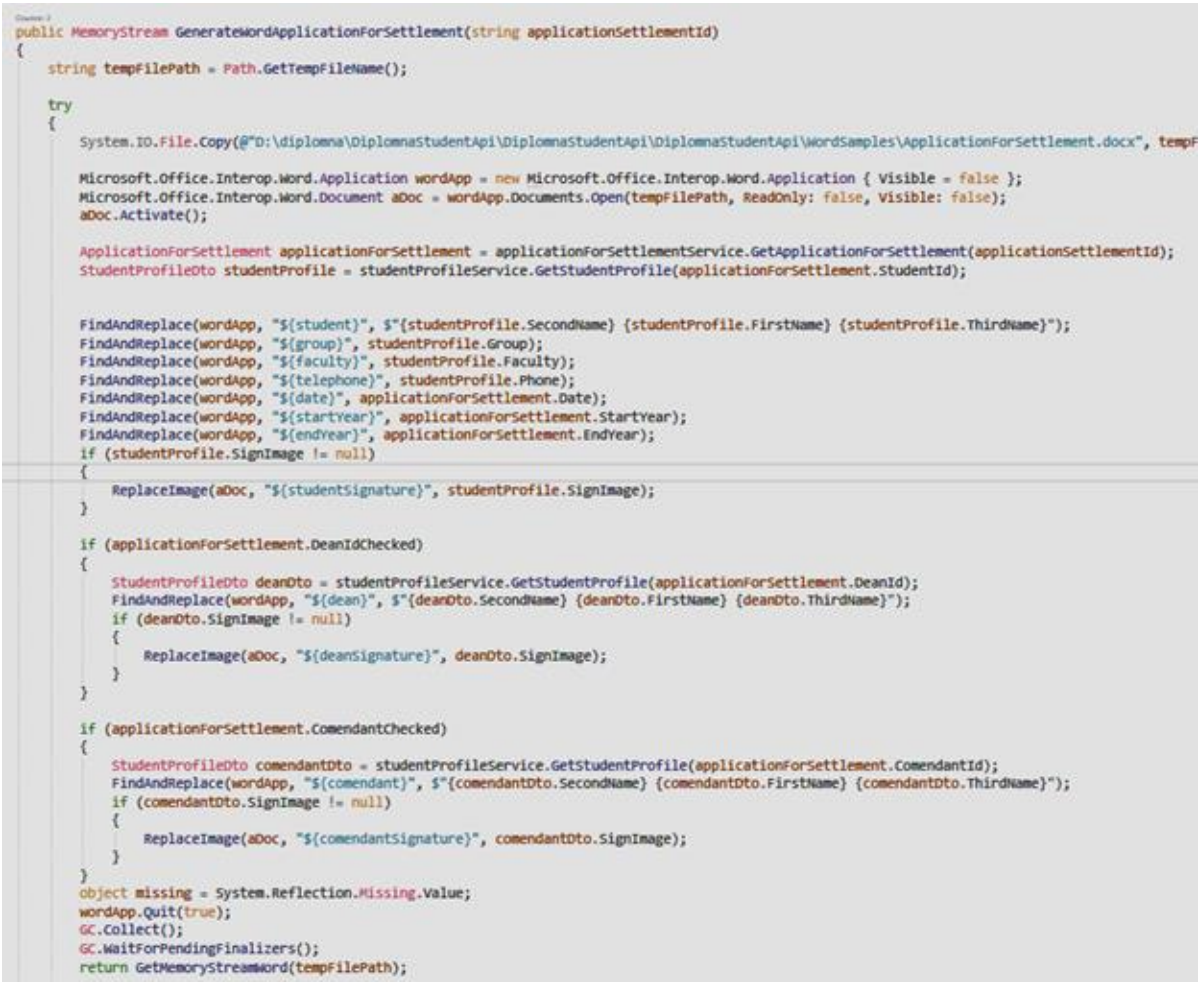
Рисунок 3.7 – Функція завантаження платежу

Також фрагмент коду завантаження платежу для фронтенда представлений на рисунку 3.8.

```
public render() {
    return (
        <div>
            <div className='blackLine'></div>
            <div className='paymentWrapper'>
                <div className='paymentLoadWrapper'>
                    <input type='file' onChange={this.handleChangePaymentCheck} />
                    <button className='save-btn' onClick={() => this.savePymentCheck()}>Зберегти</button>
                </div>
                <div className='paymentDeptWrapper'>
                    <p>{this.state.debtWord}</p>
                    <p>{this.state.debtMoney}</p>
                </div>
            </div>
            <div className='payments'>
                {this.state.paymentChecksData.map((payment, i) => {
                    // Return the element. Also pass key
                    return (<PaymendCardComponent key={i} props={payment} />)
                })}
            </div>
        </div>
    )
}
```

Рисунок 3.8 – Функція рендеренгу завантаження платежу

Також однією з найважливіших можливостей додатку для керуванням гуртожитку є можливість генерувати документи. Функції для забезпечення генерації документа представлені на рисунку 3.9.



```
public MemoryStream GenerateWordApplicationForSettlement(string applicationSettlementId)
{
    string tempFilePath = Path.GetTempFileName();

    try
    {
        System.IO.File.Copy(@"D:\diploma\DiplomnaStudentApi\DiplomnaStudentApi\DiplomnaStudentApi\WordSamples\ApplicationForSettlement.docx", tempF

        Microsoft.Office.Interop.Word.Application wordApp = new Microsoft.Office.Interop.Word.Application { Visible = false };
        Microsoft.Office.Interop.Word.Document aDoc = wordApp.Documents.Open(tempFilePath, ReadOnly: false, Visible: false);
        aDoc.Activate();

        ApplicationForSettlement applicationForSettlement = applicationForSettlementService.GetApplicationForSettlement(applicationSettlementId);
        StudentProfileDto studentProfile = studentProfileService.GetStudentProfile(applicationForSettlement.StudentId);


        FindAndReplace(wordApp, "${student}", $"{studentProfile.SecondName} {studentProfile.FirstName} {studentProfile.ThirdName}");
        FindAndReplace(wordApp, "${group}", studentProfile.Group);
        FindAndReplace(wordApp, "${faculty}", studentProfile.Faculty);
        FindAndReplace(wordApp, "${telephone}", studentProfile.Phone);
        FindAndReplace(wordApp, "${date}", applicationForSettlement.Date);
        FindAndReplace(wordApp, "${startYear}", applicationForSettlement.StartYear);
        FindAndReplace(wordApp, "${endYear}", applicationForSettlement.EndYear);
        if (studentProfile.SignImage != null)
        {
            ReplaceImage(aDoc, "${studentSignature}", studentProfile.SignImage);
        }

        if (applicationForSettlement.DeanIdChecked)
        {
            StudentProfileDto deanDto = studentProfileService.GetStudentProfile(applicationForSettlement.DeanId);
            FindAndReplace(wordApp, "${dean}", $"{deanDto.SecondName} {deanDto.FirstName} {deanDto.ThirdName}");
            if (deanDto.SignImage != null)
            {
                ReplaceImage(aDoc, "${deanSignature}", deanDto.SignImage);
            }
        }

        if (applicationForSettlement.ComendantChecked)
        {
            StudentProfileDto comendantDto = studentProfileService.GetStudentProfile(applicationForSettlement.ComendantId);
            FindAndReplace(wordApp, "${comendant}", $"{comendantDto.SecondName} {comendantDto.FirstName} {comendantDto.ThirdName}");
            if (comendantDto.SignImage != null)
            {
                ReplaceImage(aDoc, "${comendantSignature}", comendantDto.SignImage);
            }
        }
        object missing = System.Reflection.Missing.Value;
        wordApp.Quit(true);
        GC.Collect();
        GC.WaitForPendingFinalizers();
        return GetMemoryStreamWord(tempFilePath);
```

Рисунок 3.9 – Функція для створення word документу

Також для генерації документів потрібні дані. Дані для генерації документів беруться з профілів студента, коменданта, паспортиста та ін. Тому потрібно забезпечити можливість заповнення та збереження інформації для профілів студента, коменданта та ін. Забезпечення інформації про профіль користувача відбувається завдяки заповненню полів та завантаженню картинок і відсиланні цих даних на сервер. Фрагмент коду збереження профілю наведений рисунку 3.10.

```
public void UpdateStudentProfile(StudentProfileDto studentUpdateRequest, string userId)
{
    StudentProfile studentProfile = studentProfileManager.GetById(userId);
    bool isExist = true;
    if (studentProfile == null)
    {
        isExist = false;
        studentProfile = new StudentProfile();
        studentProfile.Id = userId;
    }

    studentProfile.Email = studentUpdateRequest.Email;
    studentProfile.FirstName = studentUpdateRequest.FirstName;
    studentProfile.SecondName = studentUpdateRequest.SecondName;
    studentProfile.ThirdName = studentUpdateRequest.ThirdName;
    studentProfile.Phone = studentUpdateRequest.Phone;
    studentProfile.PassportNumber = studentUpdateRequest.PassportNumber;
    studentProfile.DateBirth = studentUpdateRequest.DateBirth;
    studentProfile.Course = studentUpdateRequest.Course;
    studentProfile.PassportGivenDate = studentUpdateRequest.PassportGivenDate;
    studentProfile.Group = studentUpdateRequest.Group;
    studentProfile.Faculty = studentUpdateRequest.Faculty;
    if (isExist)
    {
        studentProfileManager.Update(userId, studentProfile);
    }
    else
    {
        studentProfileManager.Add(studentProfile);
    }
}
```

Рисунок 3.10 – Фрагмент коду збереження профілю

Також фрагмент коду збереження профілю для фронтенда представлений на рисунку 3.11.

```
    </div>
    <div className='rightBlock'>
        <div className='profileImageWrapper'>
            <img src={this.state.profileImage} />
        </div>
        <input type='file' onChange={this.handleChangeProfileImage} />

        <div className='profileImageWrapper'>
            <img src={this.state.signImage} />
        </div>
        <input type='file' onChange={this.handleChangeSignImage} />
    </div>
</div>
<div className='saveSection'>
    <button className='save-btn' onClick={() => this.saveProfile()}>Зберегти</button>
</div>
```

Рисунок 3.11 – Фрагмент коду збереження профілю для фронтенда

Також одним з найважливіших функціоналів додатку, є алгоритм нарахування боргу за поточний місяць. Фрагмент коду нарахування боргу студенту за поточний місяць наведено на рисунку 3.12.

```
static void Main(string[] args)
{

    MonthPriceManager monthPriceRepository = new MonthPriceManager();
    DebtRepository debtRepository = new DebtRepository();
    MonthPrice monthPrice = monthPriceRepository.GetAll().ToList().Find(f => f.Month.Month.Equals(DateTime.Now.Month));

    if (monthPrice == null)
    {
        return;
    }

    List<Debt> debts = debtRepository.GetAll().ToList();
    foreach(var debt in debts)
    {
        debt.MoneyDept += monthPrice.Price;
        debtRepository.Update(debt.Id, debt);
    }
}
```

Рисунок 3.12 – Фрагмент коду нарахування боргу студенту за поточний місяць

Одним з основних функціоналів є створення документу студентом на проживання у гуртожитку. Фрагмент коду створення студентом документу на проживання у гуртожитку наведено на рисунку 3.13.

```
public void GenereNewResidenceAgreement(string studentId, int studyYear)
{
    ResidenceAgreement residenceAgreement = residenceAgreementManager.GetFirstOrDefault
        (filter => (studentId.Equals(filter.StudentId) && filter.IsActive == true));
    if (residenceAgreement != null)
    {
        return;
    }

    residenceAgreement = new ResidenceAgreement();
    residenceAgreement.Id = new ObjectId(DateTime.Now, 1, 1, 1).ToString();
    residenceAgreement.IsActive = true;
    residenceAgreement.YearsSignatures = new List<ResidenceAgreementSignature>();
    residenceAgreement.LastYear = studyYear;
    residenceAgreement.StudentId = studentId;
    residenceAgreementManager.Add(residenceAgreement);
}
```

Рисунок 3.13 – Фрагмент коду створення документу на проживання у гуртожитку

Перед тим як получити можливість генерації документу потрібо декану та коменданту підтвердити заяву студента. Фрагмент коду підтвердження заяви студента деканом або комендантом наведено на рисунку 3.14.

```
public void SignResidenceAgreement(string studentId, string comendantId)
{
    ResidenceAgreement residenceAgreement = residenceAgreementManager.GetFirstOrDefault
        (filter => (studentId.Equals(filter.StudentId) && filter.IsActive == true));
    int lastYearSigned = residenceAgreement.YearsSignatures.Last().Year;
    if (lastYearSigned == residenceAgreement.LastYear)
    {
        logger.LogWarning($"Studen {studentId} is already Signed");
        return;
    }
    ResidenceAgreementSignature signature = new ResidenceAgreementSignature();
    signature.ComendantId = comendantId;
    signature.Year = residenceAgreement.LastYear;
    signature.IsAssigned = true;
    residenceAgreement.YearsSignatures.Add(signature);

    if (residenceAgreement.ComendantId == null)
    {
        residenceAgreement.ComendantId = comendantId;
    }

}
```

Рисунок 3.14 – Фрагмент коду підтвердження заяви студента

На рисунку 3.15 представлено код для мобільного додатку на фреймворці Flutter.

```
@override
Widget build(BuildContext context){
  return Scaffold(
    backgroundColor: Colors.grey,
    appBar: AppBar(
      title: Text(''),
    ),
    body: ListView.builder
      (itemBuilder: (BuildContext context, int index){
        return Dismissible(
            key: Key(notificationHeaders[index]),
            child: Card(
              child: ListTile
                (
                  title: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      Text(notificationHeaders[index]),
                      Text(notificationBody[index]),
                      Row(
                        children: <Widget>[
                          Text(notificationTime[index]),
                          Spacer(), // use Spacer
                          Text(notificationAuthor[index]),
                        ],
                      )
                    ],
```

Рисунок 3.15 – Фрагмент коду для мобільного додатку Flutter

Фрагмент коду рендерингу сторінки для менеджменту заяви на проживання наведено на рисунку 3.16.

```
public render() {
    return (
        <div className='application-setlement-card'>
            <p>{this.props.applicationSettlement.date}</p>
            <p>{this.props.applicationSettlement.startYear}-{this.props.applicationSettlement.endYear}</p>
            <div className='check-box'>
                <p>Декан</p>
                <input type="checkbox" checked={this.state.deanIdChecked} onChange={this.onChangeDeanChecked} />
            </div>
            <div className='check-box'>
                <p>Комендант</p>
                <input type="checkbox" checked={this.state.comendantChecked} onChange={this.onChangeComendantChecked} />
            </div>
            <button onClick={() => this.signDocument()} className='save-btn'>підписати</button>
            <button onClick={() => this.generateWord()} className='generete-word'>Згунерувати заяву</button>
        </div>
    )
}
```

Рисунок 3.16 – Фрагмент коду рендеренгу сторінки для менеджменту заяви на проживання

На рисунку 3.17 представлено код для рендерингу випадаючої картинки.

```
renderExpandImage() {
    if (this.state.viewerIsOpen === false) {
        return (
            <div></div>
        )
    }
    return (
        <div className='expandedImage'>
            <div className='image'>
                <TransformWrapper
                    initialScale={1}
                    initialPositionX={200}
                    initialPositionY={100}>
                    {({ zoomIn, zoomOut, resetTransform, ...rest }) => (
                        <React.Fragment>
                            <div className="tools">

                                <button className='closeButton' onClick={() => this.closeViewer()}>X</button>
                            </div>
                            <TransformComponent>
                                <img src={this.state.paymentCheck} alt="test" />
                                <div>Example text</div>
                            </TransformComponent>
                        </React.Fragment>
                    )}
                </TransformWrapper>
            </div>
        </div>
    );
}
```

Рисунок 3.17 – Фрагмент коду випадаючої картинки

Також на рисунку 3.18 наведено фрагмент коду, що відповідає за авторизацію та аутентифікацію на стороні фронтенду.

```
class AuthService {
  public userManager: UserManager;
  constructor() {
    const settings = {
      authority: 'https://localhost:5000/',
      client_id: 'oidcClient',
      redirect_uri: window.location.origin + '/signin-callback.html',
      silent_redirect_uri: window.location.origin + '/silent-renew.html',
      post_logout_redirect_uri: window.location.origin,
      response_type: process.env.REACT_APP_RESPONSE_TYPE,
      scope: 'api1.read IdentityServerApi',
      automaticSilentRenew: true,
      filterProtocolClaims: true,
      loadUserInfo: true,
      monitorSession: true,
    };
    this.userManager = new UserManager(settings);

    Log.logger = console;
    Log.level = Log.INFO;
  }

  public getUser(): Promise<User | null> {
    return this.userManager.getUser();
  }

  public login(): Promise<void> {
    return this.userManager.signinRedirect();
  }

  public renewToken(): Promise<User> {
    return this.userManager.signinSilent();
  }
```

Рисунок 3.18 – Фрагмент коду, що відповідає за авторизацію та аутентифікацію користувача

Отже, у підрозділі 3.3 було описано програмний код основних алгоритмів роботи додатку для керування гуртожитком. Повний лістинг наведено у додатку В.

3.4 Висновки

Отже, у поточному розділі було проаналізовано такі мови програмування як Kotlin, C++, C#, Python. Було вибрано мову програмування C# для реалізації додатку. Також було проаналізовані такі середовища розробки: Visual Studio, Visual Studio Code, Eclipse. Було вибрано Visual Studio як середовище розробки. Також було обрано MongoDB в якості СКБД. Крім того, було проведено опис програмної реалізації основних програмних модулів додатку для керування гуртожитком та представлено код для мобільного додатку на мові Flutter.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Аналіз методів тестування програмного забезпечення

Тестування програмного забезпечення [25] – це процес технічного дослідження, який перевіряє відповідність програмного продукту відносно вимог. Тестування у себе включає процес у якому необхідно знайти помилки у роботі програмного додатку. Також тестування оцінює якість та швидкість роботи продукту. Одними з найбільш поширених методів тестування є: динамічне та статичне тестування, тестування «білої скриньки», тестування «чорної скриньки». Статичне та динамічне тестування. Статичне тестування полягає у аналізі результатів розробки програмного продукту. Полягає у перевірці якості написаного коду, проводить контроль програми без запуску самої програми. Під час статичного тестування перевіряється вся документація, встановлюється на скільки програма відповідає заданим вимогам та критеріям. Динамічне тестування полягає у експлуатації програмного продукту. Працездатність та відповідність вимогам перевіряється виконанням різних тестів, у процесі тестування відбувається пошук та аналіз помилок у програмному забезпеченні.

Тестування «білої скриньки» [26]. У процесі тестування програмного додатку внутрішня структура програми є відомою. Перевіряються на працездатність внутрішні модулі програми та зв’язки між ними. Перевіряється коректність побудови модулів програми та правильність взаємодії між ними. Покриття тестами коду програми є основною характеристикою тестування «білої скриньки». Програму можна вважати перевіреною, якщо весь код покритий тестами і ці тести виконуються успішно і це тести включають різні сценарії виконання коду.

Тестування «чорної скриньки». У процесі тестування відомий функціонал програми. Досліджується правильність роботи фунціоналу програмного забезпечення. Основне місце де відбуваються тести це інтерфейс ПЗ. Тестування демонструє правильність виконання функцій програми, прийняття даних, вивід

даних, цілісність зовнішньої інформації. При тестуванні «чорної скриньки» ігнорується внутрішня побудова програми. Тестування дозволяє отримати набори даних які дозволяють майже повністю перевірити коректність виконання програми. Тестування «чорної скриньки» забезпечує знаходження наступних помилок: некоректність або відсутність функцій, помилка інтерфейсу, помилка у зовнішній структурі даних, помилок характеристик, помилок ініціалізації та завершення.

Після аналізу розглянутих методів тестування програмного забезпечення було вирішено використати методику «чорної скриньки», оскільки використання саме цієї методики дасть можливість протестувати функціональність роботи програмного додатку для керування гуртожитком та виявити помилки. Динамічним та статичним тестуванням не завжди можна виявити всі неполадки у роботі ПЗ.

4.2 Тестування розробленого програмного продукту

Тестування програмного додатку для керування гуртожитком за методикою «чорної скриньки» передбачає перевірку результатів виконання функцій додатку до введених даних. Тестування програмного додатку відбувається за допомогою тест-кейсів. Для тестування розробленого програмного додатку для керуванням гуртожитку було розроблено наступні тест-кейси:

Тест-кейс №1 – Реєстрація нового користувача:

1. Залогінитись під користувачом, що має права адміністратора.
2. Ввести логін.
3. Ввести пароль.
4. Ввести Роль.
5. Нажати кнопку «Register».

Очікуваним результатом даного тест-кейсу є додання запусу в базі даних про нового користувача.

Результат виконання тест-кейсу наведено на рисунку 4.1.

Id	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed	PasswordHash
00000000000000000000000	Student	STUDENT	NULL	NULL	0	AQAAAAEAACcQAAAAELpxQ2G3XqZpfwB69lx2wia0AkAMd4kEX7...
5BE86359-073C-434B-AD2D-A3932222DABE	scott	SCOTT	NULL	NULL	0	AQAAAAEAACcQAAAAEMltyYbWe1/wxlSYD0ZlpPFo8ADALM8aiD...
620ffe410000010001000001	Comendant	COMENDANT	NULL	NULL	0	AQAAAAEAACcQAAAAEBqiRuEsDVpVgQxT0Lt8vqiYMAqgsnAJ5Ql...
620ffe5a0000010001000001	Dean	DEAN	NULL	NULL	0	AQAAAAEAACcQAAAAElUGirrH2K/E0yOi9irbV9oEkSvlDuzRVrGYW...
620ffe790000010001000001	PassportHolder	PASSPORTHOLDER	NULL	NULL	0	AQAAAAEAACcQAAAAEOiia2WaaGo+VVIKF5ZaESatgaj7pbHly0D...
626d635d0000010001000001	Kovtun	KOVTUN	NULL	NULL	0	AQAAAAEAACcQAAAAEF8KKSY/mi7pEWwKYJSxHBMiAhiRRvkH...

Рисунок 4.1 – Результат виконання тест-кейсу №1

Було додано нового користувача з логіном Kovtun. Фактичний результат виконання відповідає очікуваному, отже тест-кейс №1 успішно пройдено.

Тест-кейс №2 – Формування документу на поселення:

1. Студент заповнює свій профіль.
2. Комендант заповнює свій профіль.
3. Декан заповнює свій профіль.
4. Студент створює заяву.
5. Комендант підписує заяву.
6. Декан підписує заяву.
7. Завантажити документ на поселення.

Очікуваним результатом тест-кейсу є згенерований файл заяви з всією інформацією та підписами. Результат виконання тест-кейсу №2 представлений на рисунку 4.2.

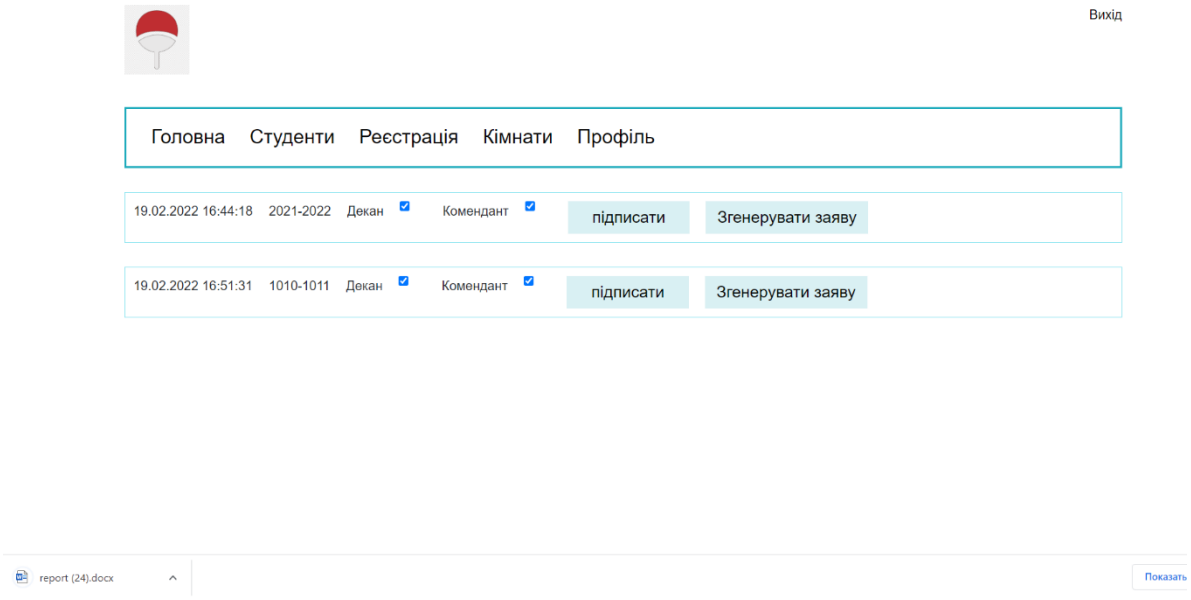


Рисунок 4.2 – Результат виконання тест-кейсу №2

В результаті виконання тест-кейсу було створено документ для поселення зі всіює потрібною інформацією. Тобто можна вважати, що тест-кейс виконаний успішно.

Тест-кейс №3 – Виконання входу в акаунт для створеного користувача:

1. Створити користувача.
2. Ввести логін користувача.
3. Ввести пароль користувача.
4. Натиснути кнопку «Login».
5. При введені неправельного логіну або паролю буде помилка.

Очікуваним результатом тест-кейсу є вхід в акаунт якщо логін та пароль введено правельно. Результат виконання тест-кейсу №3 відображено на рисунку 4.3 та на рисунку 4.4.

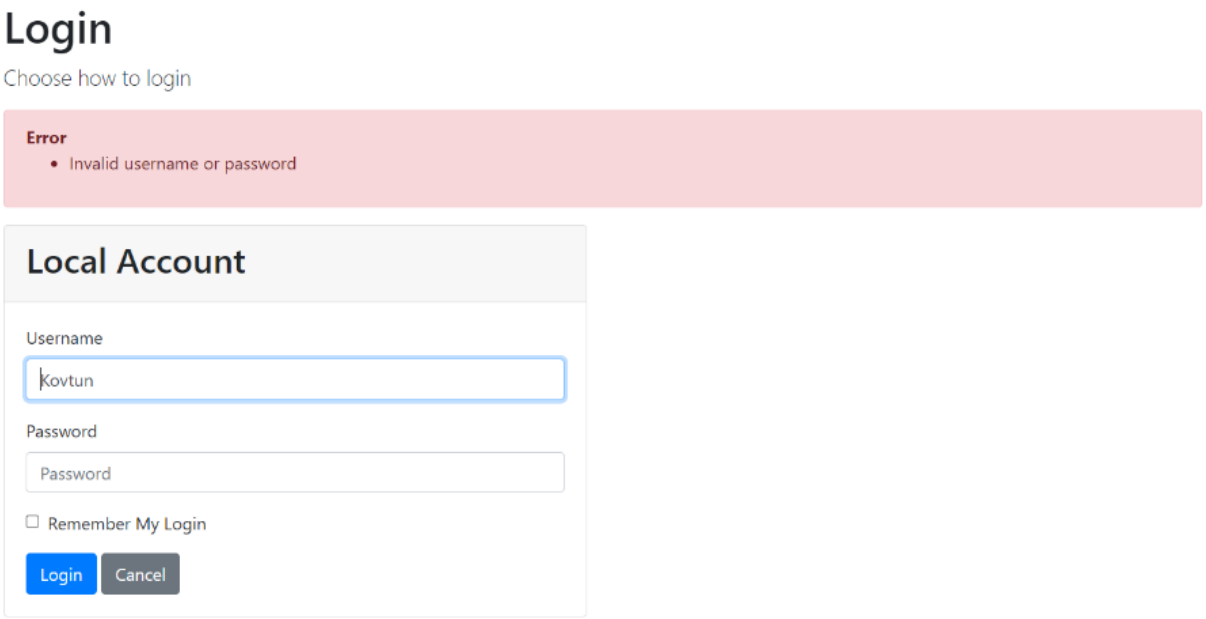


Рисунок 4.3 – Результат виконання тест-кейсу №3. При введені неправильного паролю

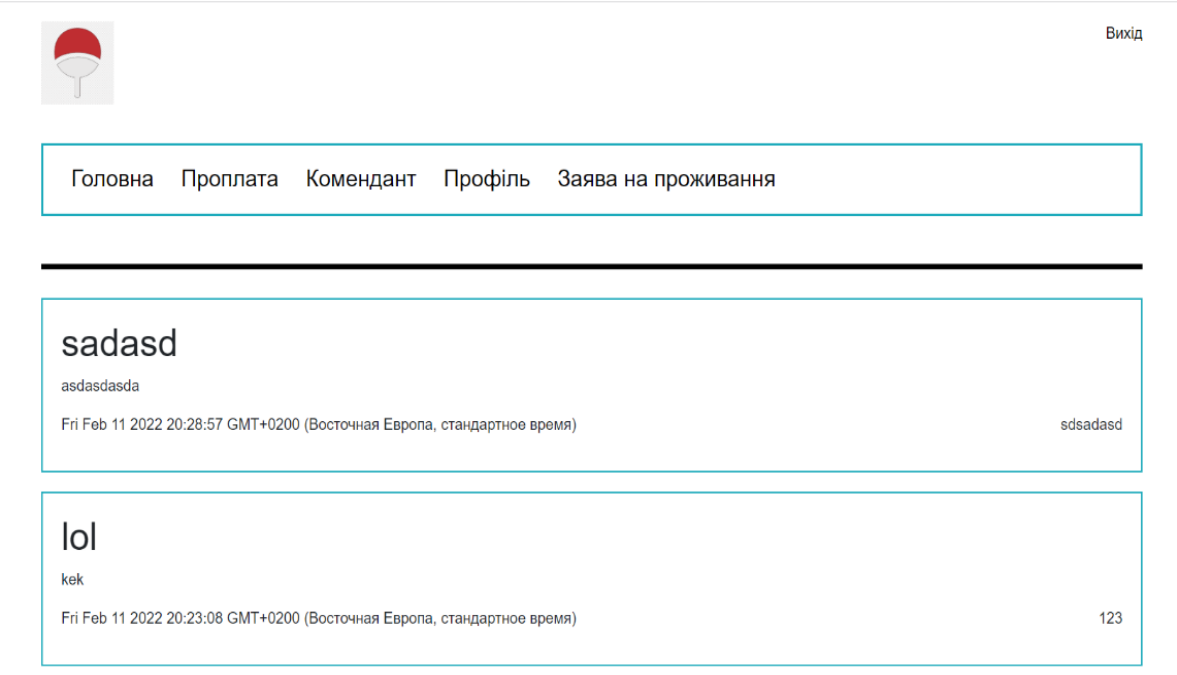


Рисунок 4.4 – Результат виконання тест-кейсу №3. При введені паролю правильно

В результаті виконання тест-кейсу було виконано логін. Також при введені неправельних даних виникає помилка. Тобто можна вважати, що тест-кейс виконаний успішно.

Тест-кейс №4 – Заповнення профілю користувача:

1. Виконати логін в акаунт.
2. Ввести ім’я.
3. Ввести прізвище.
4. Ввести по батькові.
5. Ввести телефон.
6. Ввести пошту.
7. Ввести номер паспорта.
8. Ввести групу.
9. Ввести факультет.
10. Ввести дату отримання паспорту.
11. Ввести дату народження.
12. Ввести курс навчання.

13. Загрузити аватар.
14. Загрузити підпис.
15. Зберегти зміни.

Очікуваним результатом тест-кейсу є заповнений профіль користувача. Результат виконання тест кейсу №4 наведено на рисунку 4.5.

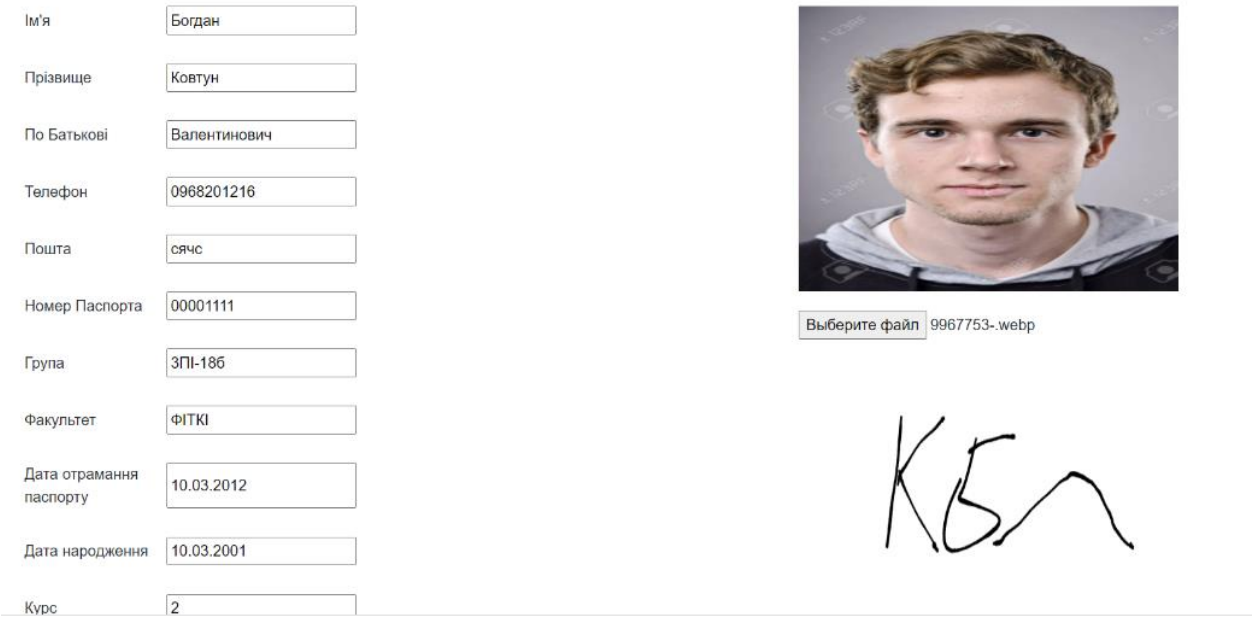


Рисунок 4.5 – Результат виконання тест-кейсу №4

В результаті виконання тест-кейсу було заповнено профіль користувача. Тобто можна вважати, що тест-кейс виконаний успішно.

Тест-кейс №5 –Завантаження квитанції про оплату:

1. Виконати вхід в акаунт.
2. Завантажити скрін кританції.
3. Натиснути кнопку «Завантажити».

Очікуваним результатом тест-кейсу є додання квитанції про оплату. Результат виконання тест-кейсу №5 представлено на рисунку 4.6.

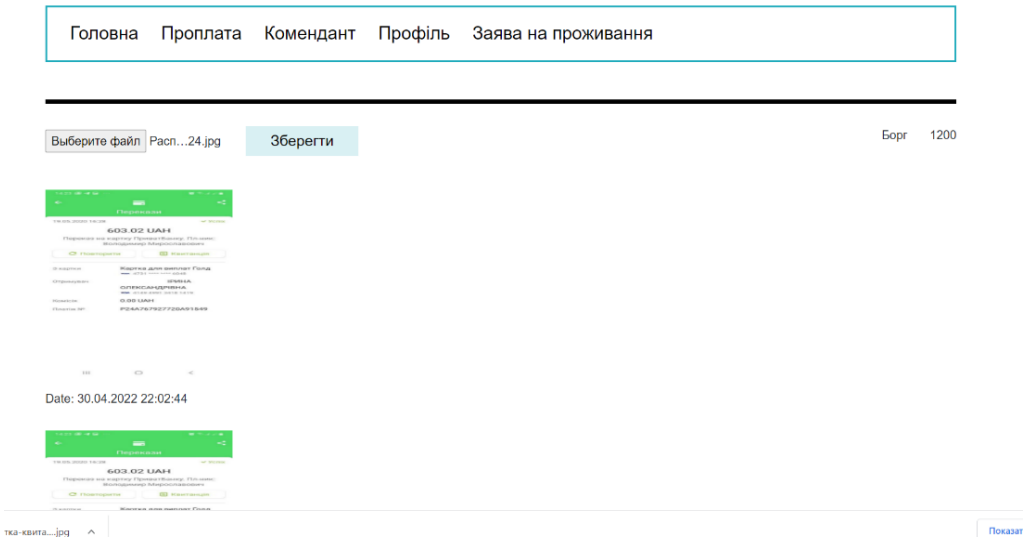


Рисунок 4.6 – Результат виконання тест-кейсу №5

В результаті виконання тест-кейсу було завантажено квитанцію про оплату. Тобто можна вважати, що тест-кейс виконаний успішно.

Тест-кейс №6 – відображення повідомлень при відкритті мобільного додатку для Flutter.

1. Завантажити додаток для телефону.
2. Відкрити додаток для телефону.

Очікуваним результатом тест-кейсу є отримання повідомлень про гуртожиток. Результат виконання тест-кейсу №6 представлено на рисунку 4.7.

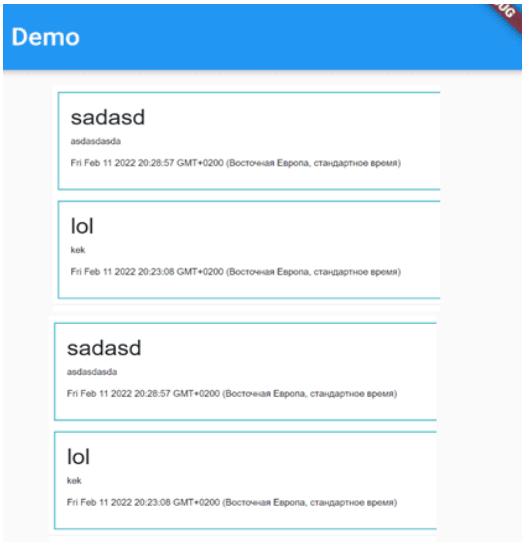


Рисунок 4.7 – Результат виконання тест-кейсу №6

В результаті виконання тест-кейсу було отримано повідомлення про гуртожиток. Тобто можна вважати, що тест-кейс виконаний успішно.

Тест кейс №7 – нарахування боргу за поточний місяць.

1. Додати ціну в базу даних за поточний місяць.
2. Запустити на виконання код для нарахування боргу.

Очікуваним результатом тест кейсу є нарахування боргу для всіх студентів. Результат виконання тест-кейсу №7 представлено на рисунку 4.8 та на рисунку 4.9.

Рисунок 4.8 – Борг студентів до виконання кроків тест-кейсу №7

Рисунок 4.9 – Борг студентів після виконання кроків тест-кейсу №7

При проведенні тестування програмного додатку не було виявлено помилок чи багів. Працездатність програмного забезпечення повністю відповідає вимогам.

4.3 Розробка інструкції користувача

Додаток для керуванням гуртожитком є веб-застосунком. Для того, щоб користуватися додатком потрібно мати профіль. В системі будуть вбудовані 4 стандартні профіля: коменданта, декана, паспортиста та адміністратора. Адміністратор має право створювати нових користувачів та присвоювати їм ролі.

Отже, для того щоб студент зміг ввійти в акаунт. Адміністратор повинен створити профіль студенту та передати йому логін та пароль. Після входу в акаунт студент буде мати можливість заповнити профіль, створити заяву на проживання, подивитися проплату, та почитати новини гуртожитку. Функціонал для студента представлений на рисунку 4.10.

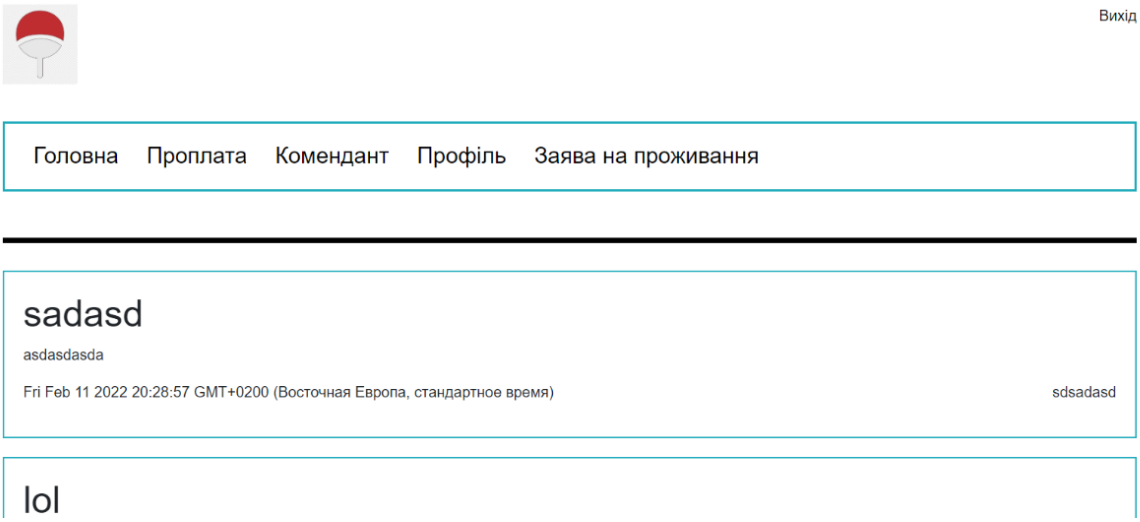


Рисунок 4.10 – Функціонал для студента

Для коменданта, паспортиста, адміністратора, декана функціональність є іншою. Наприклад адміністратор може публікувати новини для студентів та вказувати заголовок повідомлення та тіло повідомлення. Функціонал для публікації новин представлений на рисунку 4.11.

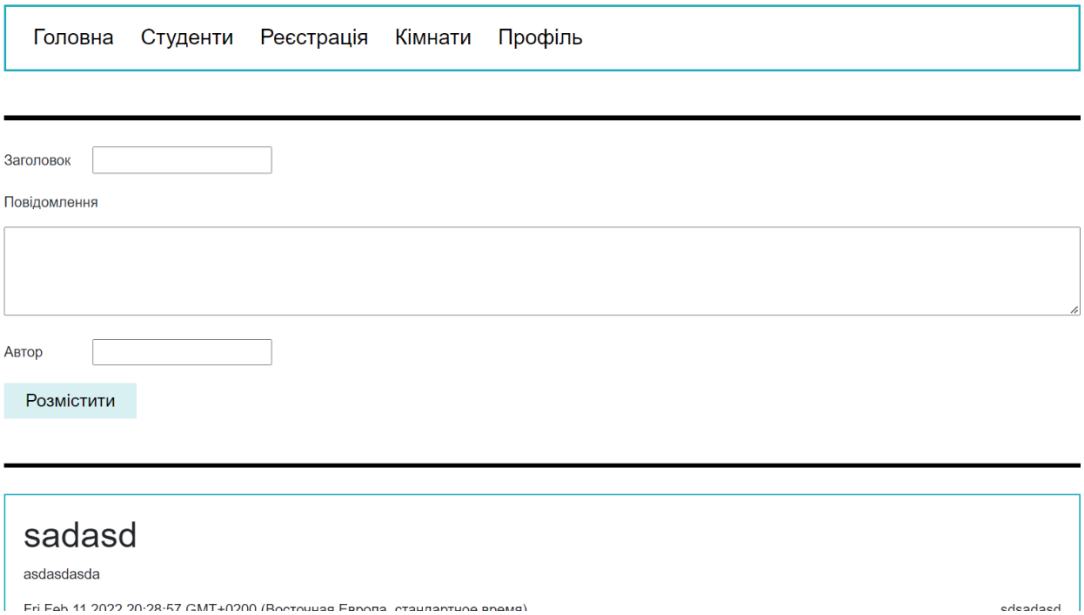


Рисунок 4.11 – Можливість адміністратора публікувати новини

Також комендант та паспортист мають панель студентів де вони можуть подивитися профілі студентів, визначити борг студента, та менеджети документи студента. Панель управління студентами представлена на рисунку 4.12.

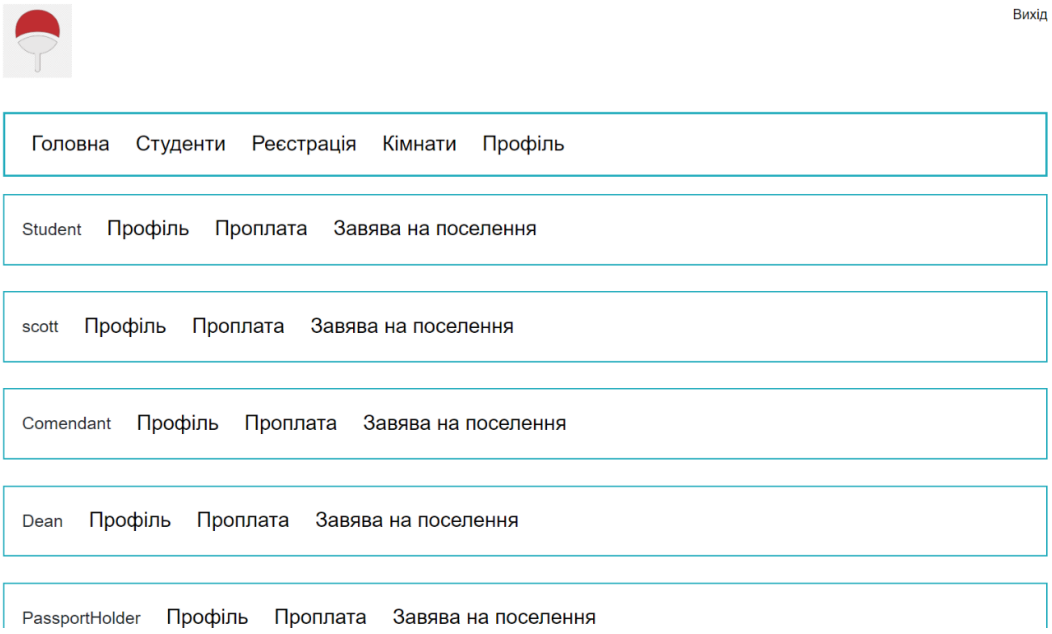


Рисунок 4.12 – Панель управління студентами

Також важливим функціоналом є створення заяви на поселення. Для того, щоб її створити та заповнити, потрібно спочатку студенту створити заяву, а потім комендант, декан ставлять помітку, що вони одобрили зяву. І після цього заяву можна згенерувати. Сторінка з менеджменту заяв студента на проживання представлена на рисунку 4.13.

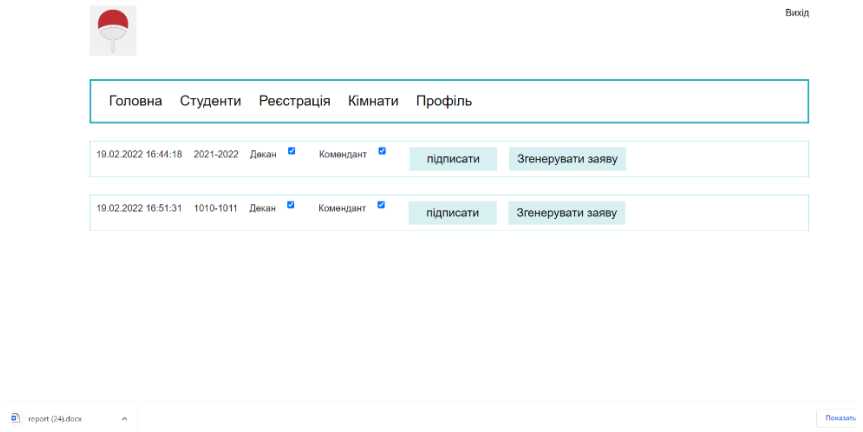


Рисунок 4.13 – Сторінка менеджменту заяв студента на проживання

Студент не має можливості зареєструватися у додатку самостійно. Акаунт студента повинен бути створений адміністратором і логін та пароль повинні бути передані студенту. Також реєстрація включає в себе вибір різних ролей. Всього існує п’ять різних ролей і це: student – студент, admin – адміністратор, comendant – комендант, dean – декан, passportHolder – паспортист. Сторінка створення нових користувачів представлена на рисунку 4.14.

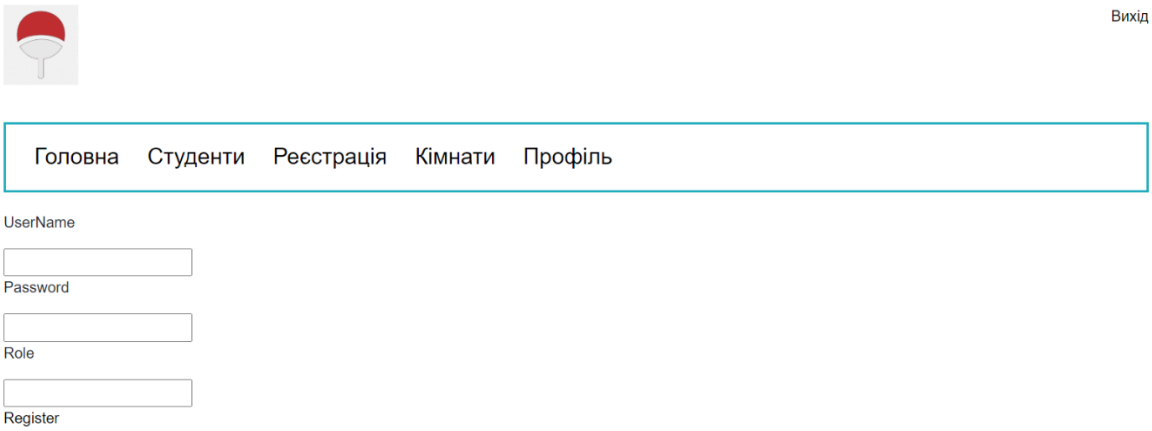


Рисунок 4.14 – Сторінка реєстрації нового користувача

Також для ефективного керування гуртожитку та списання боргу студента. Була створена можливість завантажувати квитанції. Для завантаження квитанції студенту потрібно зайти в розділ «Борг», вибрати картинку для завантаження, та натиснути кнопку «Завантажити». Потім комендант зможе подивитися цю квитанцію та списати частину боргу рівній сумі на квитанції. Сторінка для завантаження квитанції представлена на рисунку 4.15.

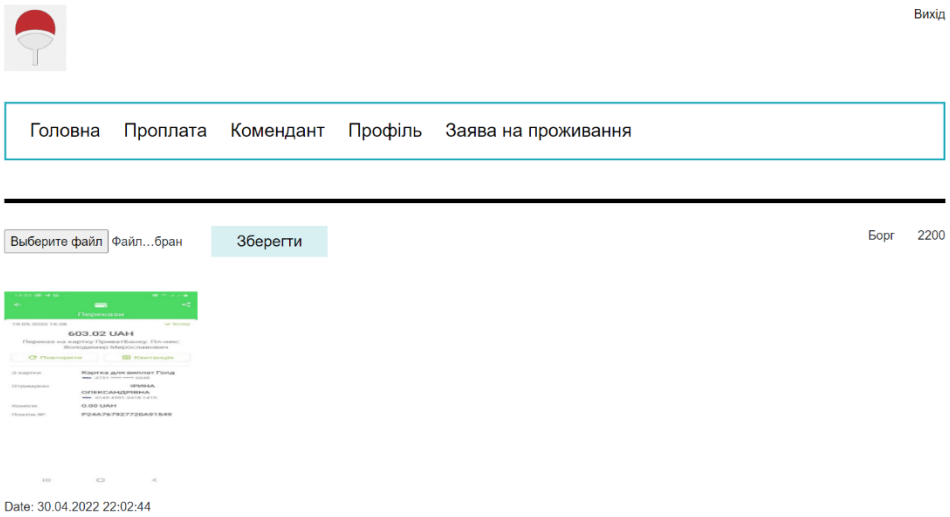


Рисунок 4.15 – Сторінка для завантаження квитанції

Отже, було розроблено інструкцію користувача. В цій інструкції описані дії для роботи з додатком для керування гуртожитком та описаний функціонал додатку для керування гуртожитком для різних типів користувачів.

4.4 Вимоги до персонального комп’ютера

На слабких машинах взаємодія користувача з додатком буде викликати труднощі. Тому додаток потрібно розробляти максимально оптимізованим. Оскільки додаток для керування гуртожитком є веб-застосунком, то для роботи з додатком потрібен лише інтернет браузер. Браузери доступні на різних платформах: Windows, Linux, Mac OS, Android. Мінімальна та рекомендована конфігурації персонального комп’ютера на базі операційної системи Windows наведено у таблицях 4.1 та 4.2.

Таблиця 4.1 – Мінімальна конфігурація

Тип процесора	64-розрядний процесор з тактовою частотою 1,8 ГГц
Оперативна пам’ять	4 ГБ
Місце на жорсткому диску	256 МБ
Відеокарта	GTX 1050
Операційна система	Windows 10

Таблиця 4.2 – Рекомендована конфігурація

Тип процесора	64-розрядний процесор з тактовою частотою 3,6 ГГц
Оперативна пам’ять	8 ГБ
Місце на жорсткому диску	256 МБ
Відеокарта	GTX 1050
Операційна система	Windows 10

Додаток не вимагає інсталяції, доступ до додатку можна получити лише відкривши сторінку у браузері.

4.5 Висновки

Для виконання тестування додатку для керуванням гуртожитку було використано метод «чорної скриньки», що означає перевірку функціональності додатку за допомогою тест-кейсів. При виконанні тестування не було виявлено багів та помилок і продукт є працездатним. Також було створено інструкцію користувача та описано функціональність додатку для різних типів користувачів. Також було визначено мінімальні та рекомендовані системні конфігурації.

ВИСНОВКИ

У процесі виконання бакалаврської дипломної роботи було розроблено веб-додаток для керування гуртожитком з використанням мультиролевої авторизації. Призначенням розробленого додатку є моніторинг боргу студентом, збереження даних студента, генерація документів для керування гуртожитком.

Було проведено аналіз аналогів додатків для керування орендою житла, було визначено їх переваги та недоліки, проведено порівняння з додатком для керування гуртожитком. В результаті було визначено доцільність розробки нового програмного додатку керування гуртожитку. Виконано постановку задач розробки програмного продукту.

Був розроблений графічний інтерфейс веб-додатку для керування гуртожитком. Запропоновано метод моніторингу боргів студента, шляхом виконання щомісячного автоматичного нарахування боргу та списання боргу при підтвердженні квитанції про сплату студентом, що дозволяє підвищити точність розрахунку боргу студента та спростити процес моніторингу боргів. Удосконалено алгоритм автоматичного генерування документів для керування гуртожитком, який дозволяє завантажувати документ лише після його підпису усіма сторонами, що дало можливість підвищити надійність та достовірність автоматично згенерованих документів.

Було проведено аналіз мов програмування, середовищ розробки, баз даних. Для розробки додатку було обрано мову C#, середовище розробки Visual Studio, систему керування базою даних MongoDB, фреймоворки ASP.NET, Flutter.

Було проаналізовано основні методики тестування. Було обрано методику «чорного ящика». При тестуванні були розроблені та перевірені тест-кейси, що довели працездатність та відповідність додатку до технічного завдання. Було розроблено інструкцію користувача та визначено вимоги до комп’ютера.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Brandon Turner. The Book on Rental Property Investing. Denver: BiggerPockets Publishing, 2015. – 347 p.

2. Yvonne Wilson. Automation for management. Gower House: Gower P. Ltd, 1969. – 266 p.

3. Colette Burrus. Developing Web Services for Web Applications. Indiana: IBM Press, 2017. – 400 p.

4. Ковтун Б.В., Романюк О.В. Використання атоматичної генерації документів для підвищення ефективності менеджменту. Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії, м. Вінниця, 30-31 травня 2022 р. Вінниця, 2022. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/author/downloadFile/16113/57352/1.

5. Ковтун Б.В., Манич А.М., Романюк О.В. Порівняльна характеристика реляційних та NoSQL баз даних. Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії, м. Вінниця, 22-23 березня 2020 р. Вінниця, 2020. URL: https://conferences.vntu.edu.ua/index.php/all-hum/all-hum-2019/paper/view/7461/6291.

6. Ковтун Б.В., Романюк О.В. Всеукраїнська науково-технічна конференція молодих вчених, аспірантів і студентів «Стан, досягнення і перспективи інформаційних систем і технологій» , м. Вінниця, 21-22 квітня 2022 р. Вінниця, 2022. URL: https://ontu.edu.ua/download/konfi/2022/Conference_abstract-IT-21-22-04-22.pdf.

7. Положення про користування гуртожитком ВНТУ. URL: https://vntu.edu.ua/uploads/2021/ kgurt.pdf.

8. HTTP protocol. IBM. URL: https://www.ibm.com/docs/en/cics-ts/5.3?topic=concepts-http-protocol

9. Rahul Sahay. ASP.NET Identity: Including ASP.NET Core. Denver: BiggerPockets Publishing, 2016. – 209 p.

10. Adam Freeman. Pro ASP.NET Core 6. Denver: Apress, 2022. –1286 p.

11. Hank Meyne. Developing Web Applications with ASP.NET. New Jersey: Wiley, 2002. – 448 p.

12. J. Stephens. Usable Web Interface Components Paperback. New York: APress, 2004. – 400 p.

13. Greg Sidelnikov. React.js Book: Learning React JavaScript Library From Scratch. Texas: River Tigris LLC, 2016. – 94 p.

14. Bas P. Harenslak. Data Pipelines with Apache Airflow. New York: Manning, 2021. – 480 p.

15. Stanley Lippman. C++ Primer. Boston: Addison-Wesley Professional, 2012. – 976 p.

16. RB Whitaker. The C# Player's Guide. London: Starbound Software, 2022. – 495 p.

17. Dmitry Jemerov. Kotlin in Action. New York: Manning, 2017. – 360 p.

18. Mark Lutz. Learning Python. Sebastopol: O'Reilly Media, 2013. – 1648 p.

19. Visual Studio. Microsoft documentation. URL: https://docs.microsoft.com/ru-ru/visualstudio/windows/?view=vs-2017

20. Bruce Johnson .Visual Studio Code: End-to-End. New Jersey: Wiley, 2019. – 192 p.

21. Eclipse. Eclipse documentation. URL: https://www.eclipse.org/documentation/.

22. MSSQL. Microsoft documentation. URL: https://docs.microsoft.com/ru-ru/sql/?view=sql-server-ver16.

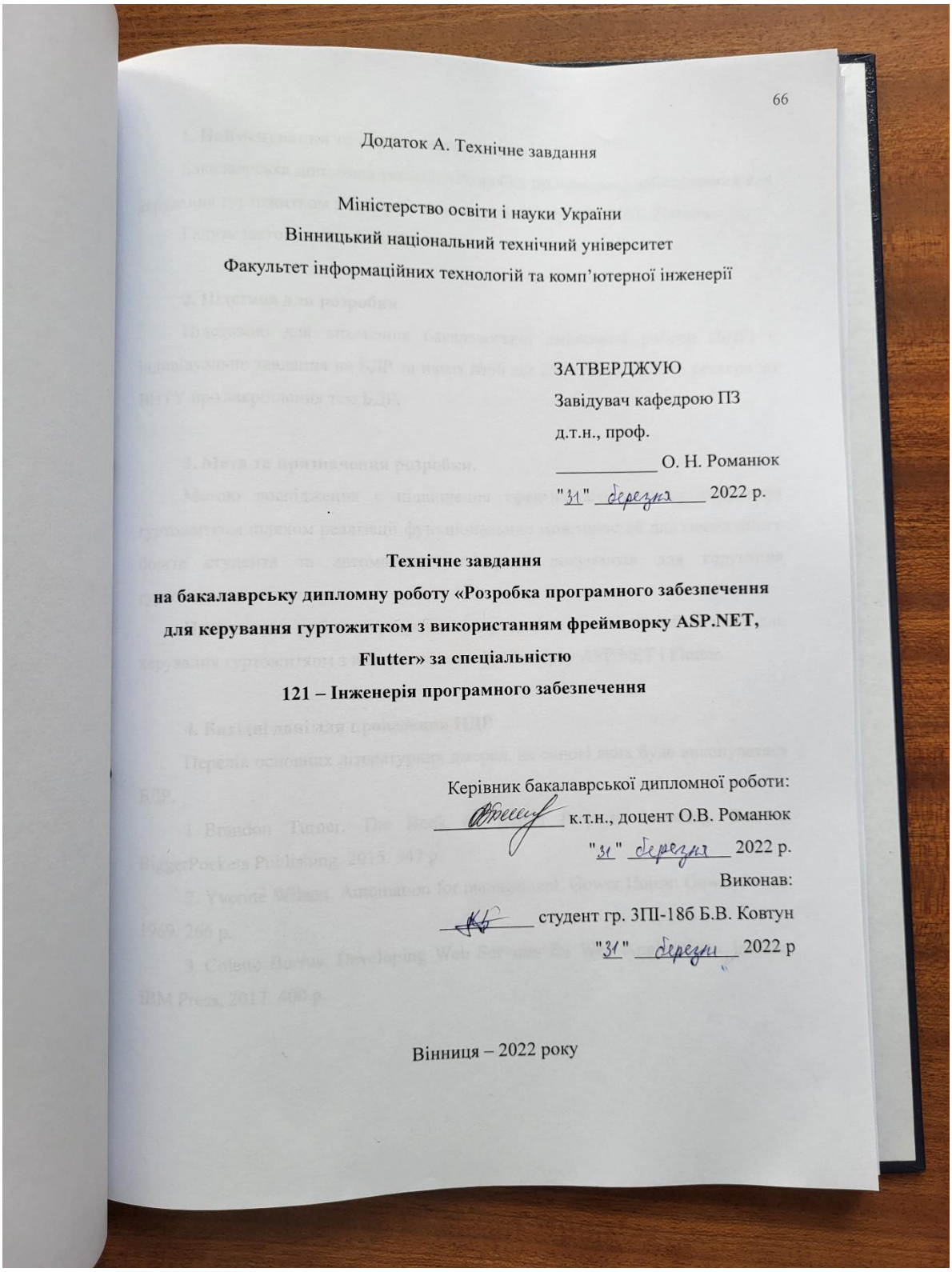
23. MySQL. MySQL documentation. URL: https://dev.mysql.com/doc/.

24. Kristina Chodorow. MongoDB: The Definitive Guide. Sebastopol: O'Reilly Media, 2013. – 432 p.

25. David Thomas. The Pragmatic Programmer. Sebastopol: O'Reilly Media, 2019. – 352 p.

26. P. RAJNI. Software Testing: Effective Methods. Sebastopol: O'Reilly Media, 2015. – 643 p.

ДОДАТКИ



Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
д.т.н., проф.
_____ О. Н. Романюк
"31" березня 2022 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка програмного забезпечення для керування гуртожитком з використанням фреймворку ASP.NET, Flutter» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ к.т.н., доцент О.В. Романюк
"31" березня 2022 р.
Виконав:
_____ студент гр. ЗПІ-18б Б.В. Ковтун
"31" березня 2022 р

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка програмного забезпечення для керування гуртожитком з використанням фреймворку ASP.NET, Flutter».

Галузь застосування – менеджмент.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №66 від 24 березня 2022 р. ректора по ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою дослідження є підвищення ефективності процесу керування гуртожитком шляхом реалізації функціональних можливостей для моніторингу боргів студента та автоматичної генерації документів для керування гуртожитком.

Призначення роботи – розробка та програмна реалізація веб-додатку для керування гуртожитком з використанням фреймворку ASP.NET і Flutter.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР.

1. Brandon Turner. The Book on Rental Property Investing. Denver: BiggerPockets Publishing, 2015. 347 p.

2. Yvonne Wilson. Automation for management. Gower House: Gower P. Ltd, 1969. 266 p.

3. Colette Burrus. Developing Web Services for Web Applications. Indiana: IBM Press, 2017. 400 p.

5. Технічні вимоги

Модель розробки – Agile; метод передачі повідомлень між серверами – HTTP; вхідні дані – інформація від користувачів, квитанція про оплату; вихідні дані – документи для керування гуртожитком; середовище розробки – VisualStudio; мова програмування – C#; фреймворк – ASP.NET, Flutter; СКБД – MongoDB.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БДР;
2. Технічне завдання;
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

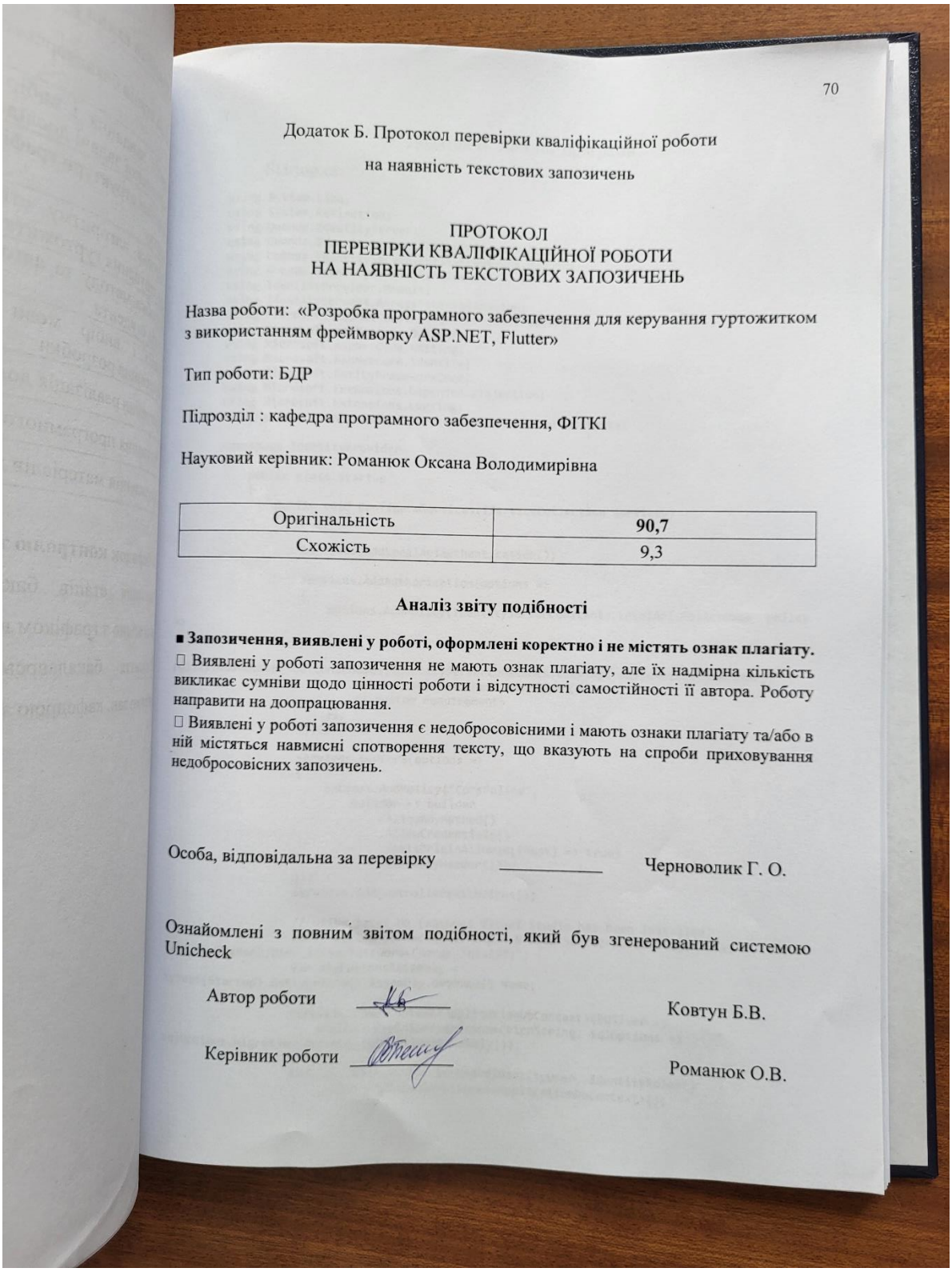
9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання і вибір методу вирішення поставленої задачі дослідження	26.03.2022 – 03.04.2022	Виконано
2	Розробка структури графічного інтерфейсу додатку	04.04.2022 – 09.04.2022	Виконано
3	Розробка алгоритму створення документів для керування гуртожитком	10.04.2022 – 17.04.2022	Виконано
4	Розробка методу та алгоритмів моніторингу боргів студента	18.04.2022 – 28.04.2022	Виконано
5	Аналіз і вибір мови програмування та середовища розробки	29.04.2022 – 03.05.2022	Виконано
6	Програмна реалізація додатку	04.05.2022 – 19.05.2022	Виконано
7	Тестування програмного забезпечення	20.05.2022 – 25.05.2022	Виконано
8	Оформлення матеріалів до захисту БДР	26.05.2022 – 10.06.2022	Виконано

10. Порядок контролю та прийняття

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.



Додаток Б. Протокол перевірки кваліфікаційної роботи
на наявність текстових запозичень

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка програмного забезпечення для керування гуртожитком з використанням фреймворку ASP.NET, Flutter»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Романюк Оксана Володимирівна

Оригінальність	**90,7**
Схожість	9,3

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Ковтун Б.В.

Керівник роботи _____ Романюк О.В.

Додаток В. Лістинг програми

Startup.cs

```
using System.Linq;
using System.Reflection;
using Duende.IdentityServer;
using Duende.IdentityServer.EntityFramework.DbContexts;
using Duende.IdentityServer.EntityFramework.Mappers;
using Duende.IdentityServer.Services;
using IdentityProvider.Models;
using IdentityServer4.AccessTokenValidation;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using static Duende.IdentityServer.IdentityServerConstants;

namespace IdentityProvider
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {

            services.AddLocalApiAuthentication();

            services.AddAuthorization(options =>
            {
                options.AddPolicy(IdentityServerConstants.LocalApi.PolicyName, policy
=>
                {

policy.AddAuthenticationSchemes(IdentityServerConstants.LocalApi.AuthenticationScheme);
                    policy.RequireAuthenticatedUser();
                    // custom requirements
                });
            });

            services.AddCors(options =>
            {
                options.AddPolicy("CorsPolicy",
                    builder => builder
                        .AllowAnyMethod()
                        .AllowCredentials()
                        .SetIsOriginAllowed((host) => true)
                        .AllowAnyHeader());
            });
            services.AddControllersWithViews();

            // using local db (assumes Visual Studio has been installed)
            const string connectionString = @"Data Source=127.0.0.1;Initial
Catalog=Bogdan3;User id=sa;Password=Change_This123;";
            var migrationsAssembly =
typeof(Startup).GetTypeInfo().Assembly.GetName().Name;

            services.AddDbContext<ApplicationDbContext>(builder =>
                builder.UseSqlServer(connectionString, sqlOptions =>
sqlOptions.MigrationsAssembly(migrationsAssembly)));

            services.AddIdentity<ExtendedIdentityUser, IdentityRole>()
                .AddEntityFrameworkStores<ApplicationDbContext>();
```

```
                var identityServerBuilder = services.AddIdentityServer(options => {
                    options.KeyManagement.Enabled = true;
                    options.IssuerUri = "https://localhost:5000";

                });

                identityServerBuilder.AddOperationalStore(options =>
                        options.ConfigureDbContext = builder =>
                            builder.UseSqlServer(connectionString, sqlOptions =>
sqlOptions.MigrationsAssembly(migrationsAssembly)))
                    .AddConfigurationStore(options =>
                        options.ConfigureDbContext = builder =>
                            builder.UseSqlServer(connectionString, sqlOptions =>
sqlOptions.MigrationsAssembly(migrationsAssembly)));

                identityServerBuilder.AddAspNetIdentity<ExtendedIdentityUser>();

            }

            public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
            {
                app.UseDeveloperExceptionPage();
                app.UseHttpsRedirection();
                //InitializeDbTestData(app);
                app.UseCors("CorsPolicy");
                app.UseStaticFiles();


                app.UseRouting();

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



                app.UseEndpoints(endpoints => endpoints.MapDefaultControllerRoute());
            }

            /// <summary>
            /// A small bootstrapping method that will run EF migrations against the
database
            /// and create your test data.
            /// </summary>
            private static void InitializeDbTestData(IApplicationBuilder app)
            {
                using (var serviceScope =
app.ApplicationServices.GetService<IServiceScopeFactory>().CreateScope())
                {

serviceScope.ServiceProvider.GetRequiredService<PersistedGrantDbContext>().Database.Migrate(
);

serviceScope.ServiceProvider.GetRequiredService<ConfigurationDbContext>().Database.Migrate()
;

serviceScope.ServiceProvider.GetRequiredService<ApplicationDbContext>().Database.Migrate();

                    var context =
serviceScope.ServiceProvider.GetRequiredService<ConfigurationDbContext>();



                        foreach (var client in Clients.Get())
```

```
                {
                    context.Clients.Add(client.ToEntity());
                }
                context.SaveChanges();


            if (!context.IdentityResources.Any())
            {
                foreach (var resource in Resources.GetIdentityResources())
                {
                    context.IdentityResources.Add(resource.ToEntity());
                }
                context.SaveChanges();
            }

            if (!context.ApiScopes.Any())
            {
                foreach (var scope in Resources.GetApiScopes())
                {
                    context.ApiScopes.Add(scope.ToEntity());
                }
                context.SaveChanges();
            }

            if (!context.ApiResources.Any())
            {
                foreach (var resource in Resources.GetApiResources())
                {
                    context.ApiResources.Add(resource.ToEntity());
                }
                context.SaveChanges();
            }

            var userManager =
serviceScope.ServiceProvider.GetRequiredService<UserManager<ExtendedIdentityUser>>();

                foreach (var testUser in Users.Get())
                {
                    var identityUser = new ExtendedIdentityUser()
                    {
                        Id = testUser.SubjectId
                    };

                    identityUser.UserName = testUser.Username;
                    identityUser.Role = "admin";

                    userManager.CreateAsync(identityUser, "Password123!").Wait();
                    userManager.AddClaimsAsync(identityUser,
testUser.Claims.ToList()).Wait();
                }

            }
        }
    }
}
```

Program.cs

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;

namespace IdentityProvider
{
    public class Program
```

```
        {
            public static void Main(string[] args)
            {
                CreateHostBuilder(args).Build().Run();
            }

            public static IHostBuilder CreateHostBuilder(string[] args) =>
                Host.CreateDefaultBuilder(args)
                    .ConfigureWebHostDefaults(webBuilder =>
                    {
                        webBuilder.UseStartup<Startup>();
                    });
        }
    }
```

Config.cs

```
    using System.Collections.Generic;
    using System.Security.Claims;
    using Duende.IdentityServer;
    using Duende.IdentityServer.Models;
    using Duende.IdentityServer.Test;
    using IdentityModel;

    namespace IdentityProvider
    {
        internal class Clients
        {
            public static IEnumerable<Client> Get()
            {
                return new List<Client>
                {

                    new Client
                    {
                        ClientId = "oidcClient",
                        ClientName = "Example Client Application",
                        //ClientSecrets = new List<Secret> {new
Secret("SuperSecretPassword".Sha256())}, // change me!

                        RequireClientSecret = false,
                        RequireConsent = false,
                        RequirePkce = true,
                         AllowedGrantTypes =  GrantTypes.Code,
                            ///AllowedCorsOrigins = { "http://localhost:3000" },
                        RedirectUris = { "http://localhost:3000/signin-callback.html",
"http://localhost:3000/refresh.html" },
                        PostLogoutRedirectUris = { "http://localhost:3000" },
                        AllowedScopes = new List<string>
                        {
                            IdentityServerConstants.StandardScopes.OpenId,
                            IdentityServerConstants.StandardScopes.Profile,
                            IdentityServerConstants.StandardScopes.Email,
                            "role",
                            "api1.read",
                            IdentityServerConstants.LocalApi.ScopeName
                        },
                    }
                };
            }
        }

        internal class Resources
        {
```

```
            public static IEnumerable<IdentityResource> GetIdentityResources()
            {
                return new[]
                {
                    new IdentityResources.OpenId(),
                    new IdentityResources.Profile(),
                    new IdentityResources.Email(),
                    new IdentityResource
                    {
                        Name = "role",
                        UserClaims = new List<string> {"role"}
                    }
                };
            }

            public static IEnumerable<ApiResource> GetApiResources()
            {
                return new[]
                {
                    new ApiResource
                    {
                        Name = "api1",
                        DisplayName = "API #1",
                        Description = "Allow the application to access API #1 on your
behalf",
                        Scopes = new List<string> {"api1.read", "api1.write"},
                        ApiSecrets = new List<Secret> {new
Secret("ScopeSecret".Sha256())}, // change me!
                        UserClaims = new List<string> {"role"}
                    },
                    new ApiResource(IdentityServerConstants.LocalApi.ScopeName),
                };
            }

            public static IEnumerable<ApiScope> GetApiScopes()
            {
                return new[]
                {
                    new ApiScope("api1.read", "Read Access to API #1"),
                    new ApiScope("api1.write", "Write Access to API #1"),
                    new ApiScope(IdentityServerConstants.LocalApi.ScopeName)
                };
            }
        }

        internal class Users
        {
            public static List<TestUser> Get()
            {
                return new List<TestUser>
                {
                    new TestUser
                    {
                        SubjectId = "5BE86359-073C-434B-AD2D-A3932222DABE",
                        Username = "scott",
                        Password = "password",
                        Claims = new List<Claim>
                        {
                            new Claim(JwtClaimTypes.Role, "admin")
                        }
                    }
                };
            }
        }
    }
```

UserController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using IdentityModel;
using IdentityProvider;
using IdentityProvider.Models;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using MongoDB.Bson;
using static Duende.IdentityServer.IdentityServerConstants;

namespace Identity.Controllers
{
    [Authorize(Policy = LocalApi.PolicyName, Roles =
"admin,comendant,dean,passportHolder")]
    [Route("[controller]")]
    public class UserController : ControllerBase
    {
        private UserManager<ExtendedIdentityUser> userManager;
        public UserController(UserManager<ExtendedIdentityUser> userManager)
        {
            this.userManager = userManager;
        }


        [HttpGet]
        [Route("/users")]
        public List<UserDto> Get()
        {
            List<UserDto> userIds = userManager.Users.Select(user => new
UserDto(user.Id, user.UserName)).ToList();
            return userIds;
        }


    }
}
```

RegistrationController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using IdentityModel;
using IdentityProvider.Models;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using MongoDB.Bson;
using static Duende.IdentityServer.IdentityServerConstants;

namespace Identity.Controllers
```

```
    {
        [Authorize(Policy = LocalApi.PolicyName, Roles = "admin")]
        [Route("[controller]")]
        public class RegistrationController : ControllerBase
        {
            private UserManager<ExtendedIdentityUser> userManager;
            public RegistrationController(UserManager<ExtendedIdentityUser> userManager)
            {
                this.userManager = userManager;
            }


            [HttpPost]
            public async Task<string> Post([FromBody] RegisterUserRequest
registerUserRequest)
            {
                bool isUserExist = await (IsUserExist(registerUserRequest));
                if (!isUserExist)
                {
                    await CreateUser(registerUserRequest);
                }
                return "success";
            }

            private async Task<bool> IsUserExist(RegisterUserRequest registerUserRequest)
            {

                ExtendedIdentityUser user = await
userManager.FindByNameAsync(registerUserRequest.UserName);
                if (user == null)
                {
                    return false;
                }
                else
                {
                    return true;
                }
            }

            private async Task CreateUser(RegisterUserRequest registerUserRequest)
            {
                var identityUser = new ExtendedIdentityUser()
                {
                    Id = new ObjectId(DateTime.Now, 1, 1, 1).ToString()
                };
                identityUser.UserName = registerUserRequest.UserName;
                var claims = new List<Claim>
                        {
                            new Claim(JwtClaimTypes.Role, registerUserRequest.Role)
                        };
                identityUser.Role = registerUserRequest.Role;
                var result = await userManager.CreateAsync(identityUser,
registerUserRequest.Password);
                result = await userManager.AddClaimsAsync(identityUser, claims);
            }
        }
    }
```

WordGeneratorServieImpl.cs

```
    using DiplomnaStudentApi.DTO;
    using DiplomnaStudentApi.Model;
    using DiplomnaStudentApi.Services;
    using System;
```

```
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Threading.Tasks;

namespace DiplomnaStudentApi.ServicesImpl
{
    public class WordGeneratorServiceImpl: IWordGeneratorService
    {
        private IApplicationForSettlementService applicationForSettlementService;
        private IStudentProfileService studentProfileService;


        public WordGeneratorServiceImpl(IApplicationForSettlementService
applicationForSettlementService, IStudentProfileService studentProfileService)
        {
            this.applicationForSettlementService = applicationForSettlementService;
            this.studentProfileService = studentProfileService;
        }


        public MemoryStream GenerateWordApplicationForSettlement(string
applicationSettlementId)
        {
            string tempFilePath = Path.GetTempFileName();

            try
            {

System.IO.File.Copy(@"D:\diplomna\DiplomnaStudentApi\DiplomnaStudentApi\DiplomnaStudentApi\W
ordSamples\ApplicationForSettlement.docx", tempFilePath, true);

                Microsoft.Office.Interop.Word.Application wordApp = new
Microsoft.Office.Interop.Word.Application { Visible = false };
                Microsoft.Office.Interop.Word.Document aDoc =
wordApp.Documents.Open(tempFilePath, ReadOnly: false, Visible: false);
                aDoc.Activate();

                ApplicationForSettlement applicationForSettlement =
applicationForSettlementService.GetApplicationForSettlement(applicationSettlementId);
                StudentProfileDto studentProfile =
studentProfileService.GetStudentProfile(applicationForSettlement.StudentId);


                FindAndReplace(wordApp, "${student}", $"{studentProfile.SecondName}
{studentProfile.FirstName} {studentProfile.ThirdName}");
                FindAndReplace(wordApp, "${group}", studentProfile.Group);
                FindAndReplace(wordApp, "${faculty}", studentProfile.Faculty);
                FindAndReplace(wordApp, "${telephone}", studentProfile.Phone);
                FindAndReplace(wordApp, "${date}", applicationForSettlement.Date);
                FindAndReplace(wordApp, "${startYear}",
applicationForSettlement.StartYear);
                FindAndReplace(wordApp, "${endYear}",
applicationForSettlement.EndYear);
                if (studentProfile.SignImage != null)
                {
                    ReplaceImage(aDoc, "${studentSignature}",
studentProfile.SignImage);
                }

                if (applicationForSettlement.DeanIdChecked)
                {
                    StudentProfileDto deanDto =
studentProfileService.GetStudentProfile(applicationForSettlement.DeanId);
```

```
                        FindAndReplace(wordApp, "${dean}", $"{deanDto.SecondName}
{deanDto.FirstName} {deanDto.ThirdName}");
                        if (deanDto.SignImage != null)
                        {
                            ReplaceImage(aDoc, "${deanSignature}", deanDto.SignImage);
                        }
                    }

                    if (applicationForSettlement.ComendantChecked)
                    {
                        StudentProfileDto comendantDto =
studentProfileService.GetStudentProfile(applicationForSettlement.ComendantId);
                        FindAndReplace(wordApp, "${comendant}",
$"{comendantDto.SecondName} {comendantDto.FirstName} {comendantDto.ThirdName}");
                        if (comendantDto.SignImage != null)
                        {
                            ReplaceImage(aDoc, "${comendantSignature}",
comendantDto.SignImage);
                        }
                    }
                    object missing = System.Reflection.Missing.Value;
                    wordApp.Quit(true);
                    GC.Collect();
                    GC.WaitForPendingFinalizers();
                    return GetMemoryStreamWord(tempFilePath);


                }catch(Exception exc)
                {
                    System.IO.File.Delete(tempFilePath);
                    throw exc;
                }

            }

            private void ReplaceImage(Microsoft.Office.Interop.Word.Document aDoc, string
key, byte[] image)
            {
                foreach (Microsoft.Office.Interop.Word.Shape s in aDoc.Shapes)
                {
                    if (s.AlternativeText.Equals(key))
                    {
                        WritePicture(s, image);
                    }
                }
            }

            private MemoryStream GetMemoryStreamWord(string tempFilePath)
            {
                var stream = new FileStream(tempFilePath, FileMode.Open);
                var newStream = new MemoryStream();
                stream.CopyTo(newStream);
                newStream.Position = 0;
                stream.Close();
                System.IO.File.Delete(tempFilePath);
                return newStream;
            }

            private void WritePicture(Microsoft.Office.Interop.Word.Shape shape, byte []
image)
            {
                string tempFilePath = Path.GetTempFileName();
                FileStream fileStream = new FileStream(tempFilePath, FileMode.Open);
                fileStream.Write(image, 0, image.Length);
                shape.Fill.UserPicture(tempFilePath);
```

```
                fileStream.Close();
                File.Delete(tempFilePath);
            }

            private static void FindAndReplace(Microsoft.Office.Interop.Word.Application
doc, object findText, object replaceWithText)
            {
                //options
                object matchCase = false;
                object matchWholeWord = true;
                object matchWildCards = false;
                object matchSoundsLike = false;
                object matchAllWordForms = false;
                object forward = true;
                object format = false;
                object matchKashida = false;
                object matchDiacritics = false;
                object matchAlefHamza = false;
                object matchControl = false;
                object read_only = false;
                object visible = true;
                object replace = 2;
                object wrap = 1;
                //execute find and replace
                doc.Selection.Find.Execute(ref findText, ref matchCase, ref
matchWholeWord,
                    ref matchWildCards, ref matchSoundsLike, ref matchAllWordForms, ref
forward, ref wrap, ref format, ref replaceWithText, ref replace,
                    ref matchKashida, ref matchDiacritics, ref matchAlefHamza, ref
matchControl);
            }
        }
    }
```

StudentProfile.cs

```
    using DiplomnaStudentApi.DTO;
    using DiplomnaStudentApi.Model;
    using DiplomnaStudentApi.Repository;
    using DiplomnaStudentApi.Services;
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Threading.Tasks;

    namespace DiplomnaStudentApi.ServicesImpl
    {
        public class StudentProfileService: IStudentProfileService
        {
            IStudentProfileManager studentProfileManager;
            public StudentProfileService(IStudentProfileManager studentProfileManager)
            {
                this.studentProfileManager = studentProfileManager;
            }

            public void UpdateStudentProfile(StudentProfileDto studentUpdateRequest,
string userId)
            {
                StudentProfile studentProfile = studentProfileManager.GetById(userId);
                bool isExist = true;
                if (studentProfile == null)
                {
                    isExist = false;
                    studentProfile = new StudentProfile();
```

```
                    studentProfile.Id = userId;
                }

                studentProfile.Email = studentUpdateRequest.Email;
                studentProfile.FirstName = studentUpdateRequest.FirstName;
                studentProfile.SecondName = studentUpdateRequest.SecondName;
                studentProfile.ThirdName = studentUpdateRequest.ThirdName;
                studentProfile.Phone = studentUpdateRequest.Phone;
                studentProfile.PassportNumber = studentUpdateRequest.PassportNumber;
                studentProfile.DateBirth = studentUpdateRequest.DateBirth;
                studentProfile.Course = studentUpdateRequest.Course;
                studentProfile.PassportGivenDate =
studentUpdateRequest.PassportGivenDate;
                studentProfile.Group = studentUpdateRequest.Group;
                studentProfile.Faculty = studentUpdateRequest.Faculty;
                if (isExist)
                {
                    studentProfileManager.Update(userId, studentProfile);
                }
                else
                {
                    studentProfileManager.Add(studentProfile);
                }
            }

            public void UpdateStudentProfileImage(byte [] blobImage, string userId)
            {
                StudentProfile studentProfile = studentProfileManager.GetById(userId);
                bool isExist = true;
                if (studentProfile == null)
                {
                    studentProfile = new StudentProfile();
                    studentProfile.Id = userId;
                    isExist = false;
                }
                studentProfile.ProfileImage = blobImage;
                if (isExist)
                {
                    studentProfileManager.Update(userId, studentProfile);
                }
                else
                {
                    studentProfileManager.Add(studentProfile);
                }
            }

            public void UpdateSignImage(byte[] blobImage, string userId)
            {
                StudentProfile studentProfile = studentProfileManager.GetById(userId);
                bool isExist = true;
                if (studentProfile == null)
                {
                    studentProfile = new StudentProfile();
                    studentProfile.Id = userId;
                    isExist = false;
                }
                studentProfile.SignImage = blobImage;
                if (isExist)
                {
                    studentProfileManager.Update(userId, studentProfile);
                }
                else
                {
                    studentProfileManager.Add(studentProfile);
                }
```

```
            }

            public StudentProfileDto GetStudentProfile(string userId)
            {
                StudentProfile studentProfile = studentProfileManager.GetById(userId);
                if (studentProfile == null)
                {
                    return new StudentProfileDto();
                }
                StudentProfileDto studentProfileDto = new StudentProfileDto();
                studentProfileDto.Email = studentProfile.Email;
                studentProfileDto.FirstName = studentProfile.FirstName;
                studentProfileDto.SecondName = studentProfile.SecondName;
                studentProfileDto.ThirdName = studentProfile.ThirdName;
                studentProfileDto.Phone = studentProfile.Phone;
                studentProfileDto.ProfileImage = studentProfile.ProfileImage;

                studentProfileDto.PassportNumber = studentProfile.PassportNumber;
                studentProfileDto.DateBirth = studentProfile.DateBirth;
                studentProfileDto.Course = studentProfile.Course;
                studentProfileDto.PassportGivenDate = studentProfile.PassportGivenDate;
                studentProfileDto.SignImage = studentProfile.SignImage;
                studentProfileDto.Group = studentProfile.Group;
                studentProfileDto.Faculty = studentProfile.Faculty;
                return studentProfileDto;
            }
        }
    }
```

RoomService.cs

```
    using DiplomnaStudentApi.DTO;
    using DiplomnaStudentApi.Model;
    using DiplomnaStudentApi.Repository;
    using DiplomnaStudentApi.Services;
    using MongoDB.Bson;
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Threading.Tasks;

    namespace DiplomnaStudentApi.ServicesImpl
    {
        public class RoomService: IRoomService
        {
            private IRoomManager roomManager;
            private IStudentProfileManager studentProfileManager;

            public RoomService(IRoomManager roomManager, IStudentProfileManager
studentProfileManager)
            {
                this.roomManager = roomManager;
                this.studentProfileManager = studentProfileManager;
            }

            public void UpdateRoom(RoomDto roomDto)
            {
                RoomUpdateValidation(roomDto, roomDto.Name);
                Room room = roomManager.GetFirstOrDefault((roomIterator) =>
roomIterator.Name.Equals(roomDto.Name));
                bool isExist = true;
                if (room == null)
                {
                    room = new Room();
```

```
                room.Id = roomDto.Name;
                isExist = false;
            }
            room.Name = roomDto.Name;
            room.Capacity = roomDto.Capacity;
            room.UserIds = roomDto.UserIds;

            if (isExist)
            {
                roomManager.Update(room.Id, room);
            }
            else
            {
                roomManager.Add(room);
            }
            UpdateStudentProfileForRoom(room);


        }

        public RoomDto GetRoom(string roomName)
        {
            Room room = roomManager.GetFirstOrDefault((r) =>
r.Name.Equals(roomName));
            if (room == null)
            {
                return new RoomDto();
            }
            RoomDto roomDto = new RoomDto();
            roomDto.Id = room.Id;
            roomDto.UserIds = room.UserIds;
            roomDto.Name = room.Name;
            roomDto.RoomImage = room.RoomImage;
            roomDto.Capacity = room.Capacity;
            return roomDto;
        }

        public List<string> GetRoomsNames()
        {
            return roomManager.GetAll().Select(r => r.Name).ToList();
        }

        public void UpdateRoomImage(byte[] roomImage, string roomName)
        {
            Room room = roomManager.GetFirstOrDefault((r) =>
r.Name.Equals(roomName));
            room.RoomImage = roomImage;
            roomManager.Update(room.Id, room);
        }

        private void UpdateStudentProfileForRoom(Room room)
        {
            foreach (var userId in room.UserIds)
            {
                StudentProfile user = studentProfileManager.GetById(userId);
                user.Room = room.Id;
                studentProfileManager.Update(user.Id, user);
            }
        }

        private void RoomUpdateValidation(RoomDto roomDto, string roomId)
        {
            if (roomDto.Capacity < roomDto.UserIds.Count)
            {
                throw new Exception("Room Capacity is too small");
```

```
                }
                foreach(var userId in roomDto.UserIds)
                {
                    StudentProfile user = studentProfileManager.GetById(userId);
                    if (user == null)
                    {
                        throw new Exception("user not exist");
                    }
                    if (user.Room != null && !user.Room.Equals(roomId))
                    {
                        throw new Exception("User already have other room");
                    }
                }
            }


        }
    }
```

ResidenceAgreementServiceImpl.cs

```
    using DiplomnaStudentApi.DTO;
    using DiplomnaStudentApi.Model;
    using DiplomnaStudentApi.Repository;
    using DiplomnaStudentApi.Services;
    using Microsoft.Extensions.Logging;
    using MongoDB.Bson;
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Threading.Tasks;

    namespace DiplomnaStudentApi.ServicesImpl
    {
        public class ResidenceAgreementServiceImpl: IResidenceAgreementService
        {
            private IResidenceAgreementManager residenceAgreementManager;
            private ILogger<ResidenceAgreementServiceImpl> logger;

            public ResidenceAgreementServiceImpl(IResidenceAgreementManager
residenceAgreementManager, ILogger<ResidenceAgreementServiceImpl> logger)
            {
                this.residenceAgreementManager = residenceAgreementManager;
                this.logger = logger;
            }

            public void GenereNewResidenceAgreement(string studentId, int studyYear)
            {
                ResidenceAgreement residenceAgreement =
residenceAgreementManager.GetFirstOrDefault(filter => (studentId.Equals(filter.StudentId) &&
filter.IsActive == true));
                if (residenceAgreement != null)
                {
                    return;
                }

                residenceAgreement = new ResidenceAgreement();
                residenceAgreement.Id = new ObjectId(DateTime.Now, 1, 1, 1).ToString();
                residenceAgreement.IsActive = true;
                residenceAgreement.YearsSignatures = new
List<ResidenceAgreementSignature>();
                residenceAgreement.LastYear = studyYear;
                residenceAgreement.StudentId = studentId;
                residenceAgreementManager.Add(residenceAgreement);
```

```
            }

            public void SignResidenceAgreement(string studentId, string comendantId)
            {
                ResidenceAgreement residenceAgreement =
residenceAgreementManager.GetFirstOrDefault(filter => (studentId.Equals(filter.StudentId) &&
filter.IsActive == true));
                int lastYearSigned = residenceAgreement.YearsSignatures.Last().Year;
                if (lastYearSigned == residenceAgreement.LastYear)
                {
                    logger.LogWarning($"Studen {studentId} is already Signed");
                    return;
                }
                ResidenceAgreementSignature signature = new
ResidenceAgreementSignature();
                signature.ComendantId = comendantId;
                signature.Year = residenceAgreement.LastYear;
                signature.IsAssigned = true;
                residenceAgreement.YearsSignatures.Add(signature);

                if (residenceAgreement.ComendantId == null)
                {
                    residenceAgreement.ComendantId = comendantId;
                }

            }

            public List<ResidenceAgreementDto> GetResidenceAgreements(string studentId)
            {
                return residenceAgreementManager.GetAll(filter =>
filter.StudentId.Equals(studentId)).Select(m=> Utils.MapperUtil.CopyAToB<ResidenceAgreement,
ResidenceAgreementDto>(m)).ToList();
            }

            public ResidenceAgreementDto GetResidenceAgreement(string studentId)
            {
                ResidenceAgreement residentAgreement =
residenceAgreementManager.GetFirstOrDefault(filter => (filter.StudentId.Equals(studentId) &&
filter.IsActive == true));
                if (residentAgreement == null)
                {
                    return new ResidenceAgreementDto();
                }
                return Utils.MapperUtil.CopyAToB<ResidenceAgreement,
ResidenceAgreementDto>(residentAgreement);
            }
        }
    }
```

PaymentService.cs

```
    using DiplomnaStudentApi.DTO;
    using DiplomnaStudentApi.Model;
    using DiplomnaStudentApi.Repository;
    using DiplomnaStudentApi.Services;
    using MongoDB.Bson;
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Threading.Tasks;
```

```
namespace DiplomnaStudentApi.ServicesImpl
{
    public class PaymentServise: IPaymentService
    {
        private IPaymentManager paymentManager;
        public PaymentServise(IPaymentManager paymentManager)
        {
            this.paymentManager = paymentManager;
        }


        public void AddPayment(PaymentDto paymentDto, string userId)
        {
            Payment payment = new Payment();
            payment.Id = new ObjectId(DateTime.Now, 1, 1, 1).ToString();
            payment.UserId = userId;
            payment.PaymentImage = paymentDto.PaymentImage;
            payment.Date = paymentDto.Date;
            paymentManager.Add(payment);
        }


        public List<PaymentDto> GetPayments(string userId)
        {
            return paymentManager.GetAll(filter =>
filter.UserId.Equals(userId)).Select(m => new PaymentDto(m.Id, m.UserId, m.PaymentImage,
m.Date)).ToList();
        }

    }
}
```

NotificationService.cs

```
using DiplomnaStudentApi.DTO;
using DiplomnaStudentApi.Model;
using DiplomnaStudentApi.Repository;
using DiplomnaStudentApi.Services;
using MongoDB.Bson;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace DiplomnaStudentApi.ServicesImpl
{
    public class NotificationService: INotificationService
    {
        INotificationManager notificationManager;
        public NotificationService(INotificationManager notificationManager)
        {
            this.notificationManager = notificationManager;
        }

        public List<NotificationDto> GetAllNotifications()
        {
            List<Notification> notifications =
notificationManager.GetAll().Reverse().ToList();
            List<NotificationDto> notificationDtos = new List<NotificationDto>();
            foreach (var notification in notifications)
            {
                NotificationDto dto = new NotificationDto();
                dto.Author = notification.Author;
                dto.Body = notification.Body;
```

```
                dto.Date = notification.Date;
                dto.Header = notification.Header;
                notificationDtos.Add(dto);
            }
            return notificationDtos;
        }

        public void AddNotification(NotificationDto notificationDto)
        {
            Notification notification = new Notification();
            notification.Author = notificationDto.Author;
            notification.Body = notificationDto.Body;
            notification.Date = notificationDto.Date;
            notification.Header = notificationDto.Header;
            notification.Id = new ObjectId(DateTime.Now, 1, 1, 1).ToString();
            notificationManager.Add(notification);
        }

    }
}
```

DebtService.cs

```
using DiplomnaStudentApi.DTO;
using DiplomnaStudentApi.Model;
using DiplomnaStudentApi.Repository;
using DiplomnaStudentApi.Services;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace DiplomnaStudentApi.ServicesImpl
{
    public class DebtService: IDebtService
    {
        private IDebtManager debtManager;
        public DebtService(IDebtManager debtManager)
        {
            this.debtManager = debtManager;
        }

        public void UpdateDebt(DebtDto debtDto, string userId)
        {
            Debt debt = debtManager.GetById(userId);
            bool isExisted = true;
            if (debt == null)
            {
                debt = new Debt();
                debt.Id = userId;
                isExisted = false;
            }

            debt.MoneyDept = debtDto.MoneyDept;
            if (isExisted)
            {
                debtManager.Update(userId, debt);
            }
            else
            {
                debtManager.Add(debt);
```

```
                }
            }

            public DebtDto GetDebtDto(string userId)
            {
                DebtDto debtDto = new DebtDto();
                Debt debt = debtManager.GetById(userId);

                if (debt == null)
                {
                    debtDto.MoneyDept = 0;
                }
                else
                {
                    debtDto.MoneyDept = debt.MoneyDept;
                }
                return debtDto;
            }
        }
    }
```

ApplicationForSettlementServiceImpl.cs

```
    using DiplomnaStudentApi.DTO;
    using DiplomnaStudentApi.Model;
    using DiplomnaStudentApi.Repository;
    using DiplomnaStudentApi.Services;
    using DiplomnaStudentApi.Utils;
    using Microsoft.Extensions.Logging;
    using MongoDB.Bson;
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Threading.Tasks;

    namespace DiplomnaStudentApi.ServicesImpl
    {
        public class ApplicationForSettlementServiceImpl :
IApplicationForSettlementService
        {
            private IApplicationForSettlementManager applicationForSettlementManager;
            private ILogger<ApplicationForSettlementServiceImpl> logger;

            public ApplicationForSettlementServiceImpl(IApplicationForSettlementManager
applicationForSettlementManager, ILogger<ApplicationForSettlementServiceImpl> logger)
            {
                this.applicationForSettlementManager = applicationForSettlementManager;
                this.logger = logger;
            }

            public void AddApplicationForSettlement(string studentId, int yearStart)
            {

                if (IsApplicationSettlementExistForStudentWithYearStart(studentId,
yearStart))
                {
                    logger.LogWarning($"Student {studentId} already exist with this year
{yearStart} in db ");
                    throw new Exception($"Student {studentId} already exist with this
year {yearStart} in db ");
                }
```

```
                ApplicationForSettlement settlement = new ApplicationForSettlement();
                settlement.Date = DateTime.Now.ToString();
                settlement.Id = new ObjectId(DateTime.Now, 1, 1, 1).ToString();
                settlement.StudentId = studentId;
                settlement.StartYear = yearStart;
                settlement.EndYear = yearStart + 1;
                applicationForSettlementManager.Add(settlement);
            }

            public void CheckApplicationForSettlementDean(string
applicationForSettlementId, string deanId)
            {
                ApplicationForSettlement applicationForSettlement =
applicationForSettlementManager.GetById(applicationForSettlementId);
                if (applicationForSettlement == null)
                {
                    logger.LogWarning($"applicationForSettlementId not exist
{applicationForSettlementId}");
                    throw new Exception($"applicationForSettlementId not exist
{applicationForSettlementId}");
                }

                applicationForSettlement.DeanIdChecked = true;
                applicationForSettlement.DeanId = deanId;
                applicationForSettlementManager.Update(applicationForSettlementId,
applicationForSettlement);
            }

            public void CheckApplicationForSettlementCommendant(string
applicationForSettlementId, string commendantId)
            {
                ApplicationForSettlement applicationForSettlement =
applicationForSettlementManager.GetById(applicationForSettlementId);
                if (applicationForSettlement == null)
                {
                    logger.LogWarning($"applicationForSettlementId not exist
{applicationForSettlementId}");
                    throw new Exception($"applicationForSettlementId not exist
{applicationForSettlementId}");
                }

                applicationForSettlement.ComendantChecked = true;
                applicationForSettlement.ComendantId = commendantId;
                applicationForSettlementManager.Update(applicationForSettlementId,
applicationForSettlement);
            }

            public List<ApplicationForSettlementDto>
GetApplicationForSettlementDtos(string studentId)
            {
                List<ApplicationForSettlementDto> ans =
applicationForSettlementManager.GetAll(filter => filter.StudentId.Equals(studentId))
                        .Select(m => MapperUtil.CopyAToB<ApplicationForSettlement,
ApplicationForSettlementDto>(m)).ToList();
                return ans;

            }

            public ApplicationForSettlement GetApplicationForSettlement(string
applicationForSettlementId)
            {
                return
applicationForSettlementManager.GetById(applicationForSettlementId);
            }
```

```
            private bool IsApplicationSettlementExistForStudentWithYearStart(string
studentId, int yearStart)
            {
                var isExist = applicationForSettlementManager.GetFirstOrDefault(filter =>
(studentId.Equals(filter.StudentId) && yearStart == filter.StartYear));
                return (isExist != null);
            }
        }
    }
```

AdminStudentProfile.component.tsx

```
import * as React from 'react';
import './adminStudentProfile.css'
import studentProfileService from '../../../services/admin/studentProfile.service';

export default class AdminStudentProfileComponent extends React.Component<any> {
  state = {
    firstName: '',
    secondName: '',
    thirdName: '',
    phone: '',
    email: '',
    profileImage: '',
    profileImageFile: '',
    userId: '',
  }

  constructor(props: any) {

    super(props)
    this.state = {
      firstName: '',
      secondName: '',
      thirdName: '',
      phone: '',
      email: '',
      profileImage: '',
      profileImageFile: '',
      userId: '',
    }
    const search = window.location.search;
```

```
    const params = new URLSearchParams(search);
    const userId = params.get('userId');
    if (userId != null) {
        this.state.userId = userId;
    }

    this.handleChangeFirstName = this.handleChangeFirstName.bind(this);
    this.handleChangeSecondName = this.handleChangeSecondName.bind(this);
    this.handleChangeThirdName = this.handleChangeThirdName.bind(this);
    this.handleChangePhone = this.handleChangePhone.bind(this);
    this.handleChangeEmail = this.handleChangeEmail.bind(this);
    this.handleChangeProfileImage = this.handleChangeProfileImage.bind(this);
    this.updateStudentProfile();
}


handleChangeFirstName(event: any) {
    this.setState({ firstName: event.target.value });
}

handleChangeSecondName(event: any) {
    this.setState({ secondName: event.target.value });
}

handleChangeThirdName(event: any) {
    this.setState({ thirdName: event.target.value });
}

handleChangePhone(event: any) {
    this.setState({ phone: event.target.value });
}

handleChangeEmail(event: any) {
    this.setState({ email: event.target.value });
}

updateStudentProfile() {
    studentProfileService.getStudentProfile(this.state.userId).then((result) => {
        this.setState({ profileImage: `${result.data.profileImage}` });
        this.setState({ firstName: result.data.firstName || '' });
        this.setState({ secondName: result.data.secondName || '' });
        this.setState({ thirdName: result.data.thirdName || '' });
```

```
        this.setState({ phone: result.data.phone || '' });
        this.setState({ email: result.data.email || '' });
    });
}

async handleChangeProfileImage(event: any) {
    await this.setState({ profileImageFile: event.target.files[0] })

}

async saveProfileImage() {
    await studentProfileService.saveStudentProfileImage(this.state.profileImageFile, this.state.userId);

}

async saveProfileData() {
    const profileUpdateRequest = {
        FirstName: this.state.firstName,
        SecondName: this.state.secondName,
        ThirdName: this.state.thirdName,
        Phone: this.state.phone,
        Email: this.state.email,
    }

    await studentProfileService.saveStudentProfileData(profileUpdateRequest, this.state.userId);
}

async saveProfile() {
    await this.saveProfileImage();
    await this.saveProfileData();
    await this.updateStudentProfile();
}

public render() {
    return (
        <div>
            <div className='blackLine'></div>
            <div className='prifileWrapper'>
                <div className='leftBlock'>
                    <div className='controll-input'>
                        <p>Ім'я</p>
```

```
<input type="text" value={this.state.firstName}
onChange={this.handleChangeFirstName} />
</div>

<div className='controll-input'>
<p>Прізвище</p>
<input type="text" value={this.state.secondName}
onChange={this.handleChangeSecondName} />
```

DebtCalculator.cs

```
MonthPriceManager monthPriceRepository = new MonthPriceManager();
DebtRepository debtRepository = new DebtRepository();
MonthPrice monthPrice = monthPriceRepository.GetFirstOrDefault(filter =>
filter.Month.Equals(DateTime.Now.Month));
if (monthPrice == null)
{
    monthPrice = new MonthPrice();
    monthPrice.Price = 0;
}

List<Debt> debts = debtRepository.GetAll().ToList();
foreach(var debt in debts)
{
    debt.MoneyDept += monthPrice.Price;
    debtRepository.Update(debt.Id, debt);
 }
```

AdminDeptController.cs

```
using DiplomnaStudentApi.DTO;
using DiplomnaStudentApi.Model;
using DiplomnaStudentApi.Repository;
using DiplomnaStudentApi.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace DiplomnaStudentApi.Admin.Controller
{

    [ApiController]
    [Route("[controller]")]
    public class AdminDeptController : ControllerBase
    {
        private IDebtService debtService;
        public AdminDeptController(IDebtService debtService)
        {
            this.debtService = debtService;
        }

        [Authorize(Roles = "admin,comendant")]
        [Route("/admin/debt")]
        [HttpPost]
        public string UpdateDebt([FromBody] DebtDto debtDto, [FromQuery(Name = "userId")]
string userId)
        {

            debtService.UpdateDebt(debtDto, userId);
            return "ok";
```

```
        }

        [Authorize(Roles = "admin,comendant")]
        [Route("/admin/debt")]
        [HttpGet]
        public DebtDto GetDebt([FromQuery(Name = "userId")] string userId)
        {

            return debtService.GetDebtDto(userId);
        }



    }
}
```

AdminHomeController.cs

```
using DiplomnaStudentApi.DTO;
using DiplomnaStudentApi.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace DiplomnaStudentApi.Admin.Controller
{

    [ApiController]
    [Route("[controller]")]
    public class AdminHomeController : ControllerBase
    {
        private INotificationService notificationService;

        public AdminHomeController(INotificationService notificationService)
        {
            this.notificationService = notificationService;
        }

        [Authorize(Roles = "admin")]
        [Route("/home")]
        [HttpPost]
        public string AddNotification([FromBody] NotificationDto notificationDto)
        {
            notificationService.AddNotification(notificationDto);
            return "ok";

        }



    }
}
```

AdminPaymentController.cs

```
using DiplomnaStudentApi.DTO;
using DiplomnaStudentApi.Services;
using DiplomnaStudentApi.Student.Controller;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

```
namespace DiplomnaStudentApi.Admin.Controller
{
    public class AdminPaymentController : ApiController
    {
        private IPaymentService paymentService;
        public AdminPaymentController(IPaymentService paymentService)
        {
            this.paymentService = paymentService;
        }

        [Authorize(Roles = "admin")]
        [Route("admin/payment")]
        [HttpGet]
        public List<PaymentDto> GetPayments([FromQuery(Name = "userId")] string userId)
        {
            return paymentService.GetPayments(userId);
        }
    }
}
```

AdminStudentProfileController.cs

```
using DiplomnaStudentApi.DTO;
using DiplomnaStudentApi.Model;
using DiplomnaStudentApi.Repository;
using DiplomnaStudentApi.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Microsoft.Web.Helpers;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;

namespace DiplomnaStudentApi.Student.Controller
{

    [ApiController]
    [Route("[controller]")]
    public class AdminStudentProfileController : ApiController
    {
        private IStudentProfileService studentProfileService;

        public AdminStudentProfileController(IStudentProfileService studentProfileService)
        {
            this.studentProfileService = studentProfileService;
        }

        [Authorize(Roles = "admin,comendant,dean,passportHolder")]
        [Route("/admin/student/profile")]
        [HttpGet]
        public StudentProfileDto StudentProfile([FromQuery(Name = "userId")] string userId)
        {
            StudentProfileDto studentProfileDto =
studentProfileService.GetStudentProfile(userId);
            return studentProfileDto;
        }

        [Authorize(Roles = "admin,comendant,dean")]
        [Route("/admin/student/profileImage/update")]
        [HttpPost]
```

```
        [RequestSizeLimit(40000000)]
        public string UpdateProfileImage([FromQuery(Name = "userId")] string userId)
        {
            byte[] blobImage = base.GetImageFromRequestBytes();
            studentProfileService.UpdateStudentProfileImage(blobImage, userId);
            return "ok";

        }


        [Authorize(Roles = "admin,comendant,dean")]
        [Route("/admin/student/profile/update")]
        [HttpPost]
        public string UpdateProfile([FromBody] StudentProfileDto studentUpdateRequest,
[FromQuery(Name = "userId")] string userId)
        {
            studentProfileService.UpdateStudentProfile(studentUpdateRequest, userId);
            return "ok";
        }

    }
}
```

ApplicationForSettlementController.cs

```
using DiplomnaStudentApi.DTO;
using DiplomnaStudentApi.Services;
using DiplomnaStudentApi.Student.Controller;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Net.Http.Headers;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Threading.Tasks;

namespace DiplomnaStudentApi.Controllers.Student
{
    [Route("[controller]")]
    public class ApplicationForSettlementController : ApiController
    {
        private IApplicationForSettlementService applicationForSettlementService;
        private IWordGeneratorService wordGeneratorService;

        public ApplicationForSettlementController(IApplicationForSettlementService
applicationForSettlementService, IWordGeneratorService wordGeneratorService)
        {
            this.applicationForSettlementService = applicationForSettlementService;
            this.wordGeneratorService = wordGeneratorService;
        }

        [Authorize(Roles = "student")]
        [Route("/student/applicationSettlement")]
        [HttpPost]
        public string AddSettlement([FromBody] ApplicationForSettlementDto
applicationForSettlementDto)
        {
            applicationForSettlementService.AddApplicationForSettlement(GetUserId(),
applicationForSettlementDto.StartYear);
            return "ok";
        }

        [Authorize(Roles = "admin,comendant,dean,passportHolder")]
```

```
        [Route("/common/applicationSettlement")]
        [HttpGet]
        public List<ApplicationForSettlementDto> GetApplicationSettlement([FromQuery(Name =
"studentId")] string studentId)
        {
            return
applicationForSettlementService.GetApplicationForSettlementDtos(studentId);
        }

        [Authorize(Roles = "student")]
        [Route("/student/applicationSettlement")]
        [HttpGet]
        public List<ApplicationForSettlementDto> GetApplicationSettlementStudent()
        {
            return
applicationForSettlementService.GetApplicationForSettlementDtos(GetUserId());
        }

        [Authorize(Roles = "dean")]
        [Route("/dean/applicationSettlement/sign")]
        [HttpPost]
        public string DeanSignApplicationSettlement([FromQuery(Name =
"applicationSettlementId")] string applicationSettlementId)
        {

applicationForSettlementService.CheckApplicationForSettlementDean(applicationSettlementId,
GetUserId());
            return "ok";
        }

        [Authorize(Roles = "comendant")]
        [Route("/comendant/applicationSettlement/sign")]
        [HttpPost]
        public string ComendantSignApplicationSettlement([FromQuery(Name =
"applicationSettlementId")] string applicationSettlementId)
        {

applicationForSettlementService.CheckApplicationForSettlementCommendant(applicationSettlemen
tId, GetUserId());
            return "ok";
        }

        [Route("/common/generateWord")]
        [HttpGet]
        public FileStreamResult GenereteWord([FromQuery(Name = "applicationSettlementId")]
string applicationSettlementId)
        {

            MemoryStream memoryStream =
wordGeneratorService.GenerateWordApplicationForSettlement(applicationSettlementId);
            var file = File(memoryStream, "application/octet-stream", "myfile.docx");
            return file;

        }
    }
}
```

SelfProfileController.cs

```
using DiplomnaStudentApi.DTO;
using DiplomnaStudentApi.Model;
using DiplomnaStudentApi.Repository;
using DiplomnaStudentApi.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
```

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Microsoft.Web.Helpers;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;

namespace DiplomnaStudentApi.Student.Controller
{

    [ApiController]
    [Route("[controller]")]
    public class SelfProfileController : ApiController
    {
        private IStudentProfileService studentProfileService;

        public SelfProfileController(IStudentProfileService studentProfileService)
        {
            this.studentProfileService = studentProfileService;
        }


        [Authorize(Roles = "student,admin,comendant,dean,passportHolder")]
        [Route("/common/profile")]
        [HttpGet]
        public StudentProfileDto StudentProfile()
        {
            string userId = base.GetUserId();
            StudentProfileDto studentProfileDto =
studentProfileService.GetStudentProfile(userId);
            return studentProfileDto;
        }

        [Authorize(Roles = "student,admin,comendant,dean,passportHolder")]
        [Route("/common/profileImage/update")]
        [HttpPost]
        [RequestSizeLimit(40000000)]
        public string UpdateProfileImage()
        {
            string userId = base.GetUserId();
            byte[] blobImage = base.GetImageFromRequestBytes();
            studentProfileService.UpdateStudentProfileImage(blobImage, userId);
            return "ok";

        }

        [Authorize(Roles = "student,admin,comendant,dean,passportHolder")]
        [Route("/common/signImage/update")]
        [HttpPost]
        [RequestSizeLimit(40000000)]
        public string UpdateSignImage()
        {
            string userId = base.GetUserId();
            byte[] blobImage = base.GetImageFromRequestBytes();
            studentProfileService.UpdateSignImage(blobImage, userId);
            return "ok";

        }

        [Authorize(Roles = "student,admin,comendant,dean,passportHolder")]
        [Route("/common/profile/update")]
        [HttpPost]
```

```
        public string UpdateProfile([FromBody] StudentProfileDto studentUpdateRequest)
        {
            string userId = base.GetUserId();
            studentProfileService.UpdateStudentProfile(studentUpdateRequest, userId);
            return "ok";
        }

    }
}
```

StudentDebtController.cs

```
using DiplomnaStudentApi.DTO;
using DiplomnaStudentApi.Model;
using DiplomnaStudentApi.Repository;
using DiplomnaStudentApi.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace DiplomnaStudentApi.Student.Controller
{

    [Route("[controller]")]
    public class StudentDeptController : ApiController
    {
        private IDebtService debtService;
        public StudentDeptController(IDebtService debtService)
        {
            this.debtService = debtService;
        }

        [Authorize(Roles = "student")]
        [Route("/student/debt")]
        [HttpGet]
        public DebtDto GetDebt()
        {
            string userId = base.GetUserId();
            return debtService.GetDebtDto(userId);
        }



    }
}
```

StudentPaymentController.cs

```
using DiplomnaStudentApi.DTO;
using DiplomnaStudentApi.Services;
using DiplomnaStudentApi.Student.Controller;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace DiplomnaStudentApi.Controllers.Student
{
    public class StudentPaymentController: ApiController
    {
        private IPaymentService paymentService;
```

```
        public StudentPaymentController(IPaymentService paymentService)
        {
            this.paymentService = paymentService;
        }

        [Authorize(Roles = "student")]
        [Route("student/payment")]
        [HttpPost]
        public string AddPayment()
        {
            string userId = base.GetUserId();
            byte[] paymentImage = base.GetImageFromRequestBytes();
            PaymentDto paymentDto = new PaymentDto();
            paymentDto.Date = DateTime.Now.ToString();
            paymentDto.PaymentImage = paymentImage;
            paymentDto.UserId = userId;
            paymentService.AddPayment(paymentDto, userId);
            return "ok";
        }

        [Authorize(Roles = "student")]
        [Route("student/payment")]
        [HttpGet]
        public List<PaymentDto> GetPayments()
        {
            string userId = base.GetUserId();
            return paymentService.GetPayments(userId);
        }
    }
}
```

ApplicationForSettlement.tsx

```
async generateWord() {
        if (this.props.applicationSettlement.deanIdChecked &&
this.props.applicationSettlement.comendantChecked) {
            await applicationSettlementService.getWordForSettlement(this.props.applicationSettlement.id);
        } else {
            console.log('File not assigned by all users');
        }

    }

    public render() {
        return (
            <div className='application-setlement-card'>
                <p>{this.props.applicationSettlement.date}</p>
                <p>{this.props.applicationSettlement.startYear}-
{this.props.applicationSettlement.endYear}</p>
                <div className='check-box'>
                    <p>Декан</p>
                    <input type="checkbox" checked={this.state.deanIdChecked}
onChange={this.onChangeDeanChecked} />
                </div>
                <div className='check-box'>
                    <p>Комендант</p>
                    <input type="checkbox" checked={this.state.comendantChecked}
onChange={this.onChangeComendantChecked} />
                </div>
                <button onClick={() => this.signDocument()} className='save-btn'>підписати</button>
                <button onClick={() => this.generateWord()} className='generete-word'>Згенерувати
заяву</button>
            </div>
        )
    }
```

Додаток Г. Графічна частина

ГРАФІЧНА ЧАСТИНА

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ КЕРУВАННЯ ГУРТОЖИТКОМ З ВИКОРИСТАННЯМ ФРЕЙМВОРКУ ASP.NET, FLUTTER

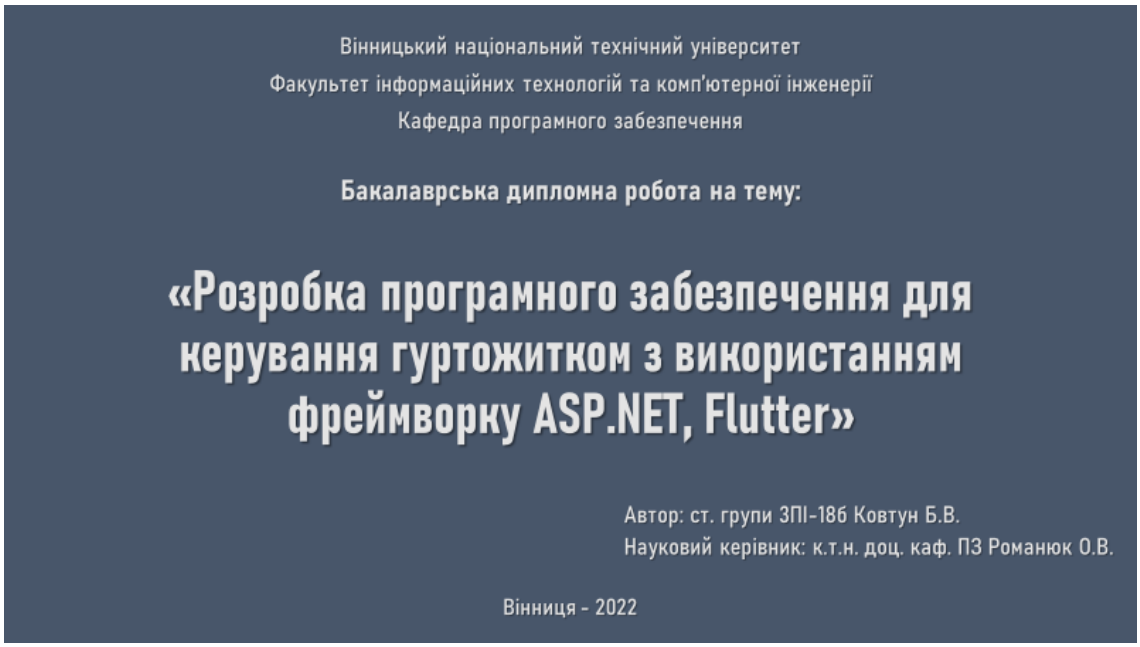


Рисунок Г.1 – Титульний слайд

Рисунок Г.2 – Актуальність теми

Рисунок Г.3 – Мета, об'єкт та предмет дослідження

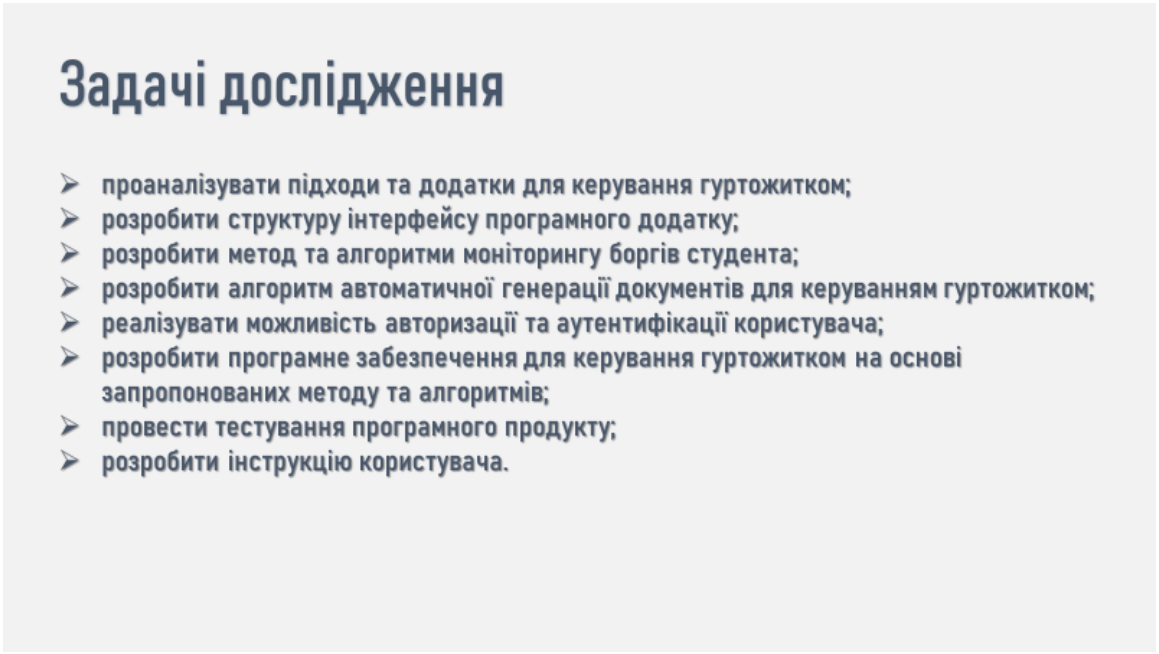


Рисунок Г.4 – Задачі дослідження

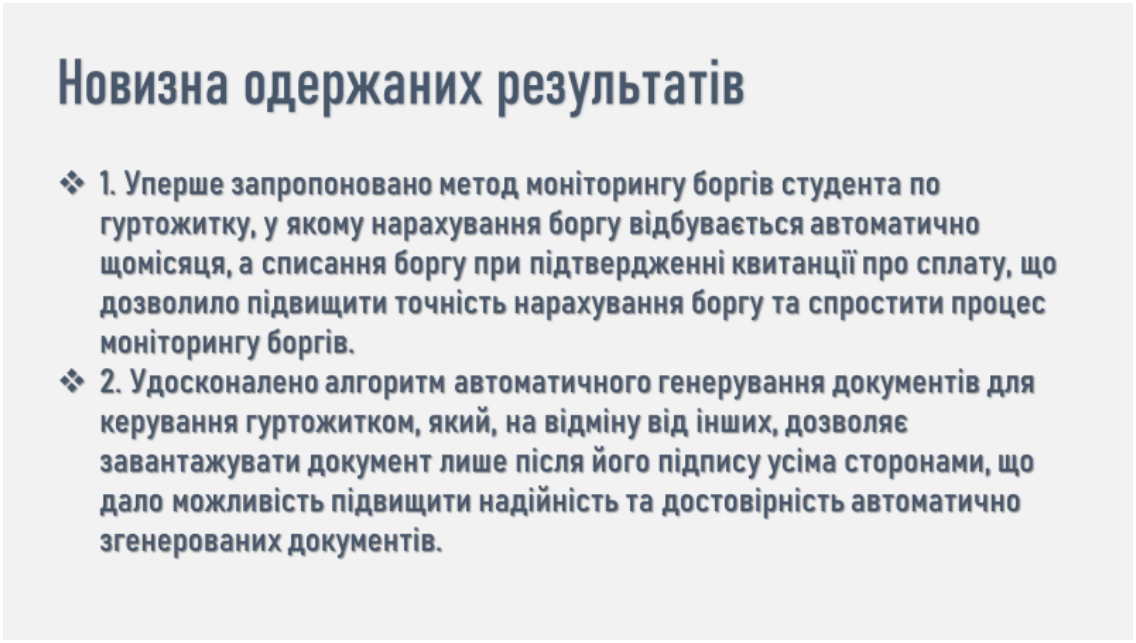


Рисунок Г.5 – Новизна одержаних результатів

Рисунок Г.6 – Практична цінність одержаних результатів

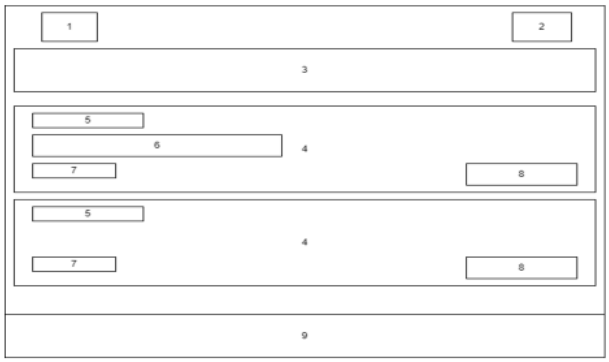
Порівняльний аналіз аналогів

Критерій	Booking.com	Aiosell	Cosmo	CloudBeds	Додаток для управління гуртожитком
Можливість створення профілю користувача	1	0	0	0	1
Генерація документів	0	0	1	1	1
Автоматичне нарахування боргу	0	0	0	0	1
Можливість публікації новини користувачам	1	1	1	1	1
Менеджмент користувачів	1	0	0	0	1
Підсумковий результат	**3**	**1**	**2**	**2**	**5**

Рисунок Г.7 – Порівняльний аналіз аналогів

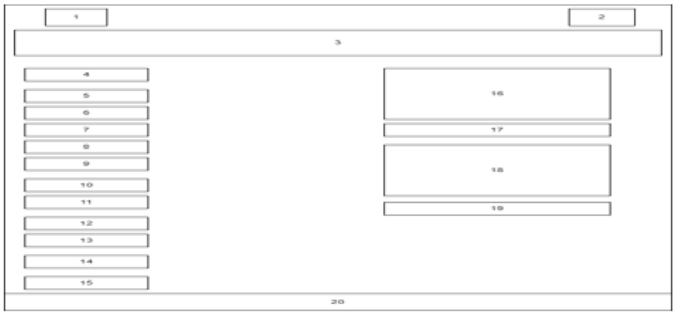
Структура графічного інтерфейсу

Головне вікно додатку



1.Лого.
2.Кнопки керування входу та виходу.
3.Меню.
4.Карта з повідомненням
5.Заголовок повідомлення.
6.Текст повідомлення.
7.Час повідомлення.
8.Автор повідомлення.
9.Інформація про авторів сайту.

Рисунок Г.8 – Структура графічного інтерфейсу головного вікна додатку



1. Лого.
2. Кнопки керування входу та виходу.
3. Меню.
4. Поле вводу ім'я.
5. Поле вводу прізвища.
6. Поле вводу по батькові.
7. Поле вводу телефону.
8. Поле вводу пошти.
9. Поле вводу номеру паспорту.
10. Поле вводу групи
11. Поле вводу факультету.
12. Поле вводу Дати отримання паспорту.
13. Поле вводу дати народження.
14. Поле вводу курсу.
15. Кнопка «Зберегти».
16. Фотографія студента.
17. Кнопка збереження фотографії студента.
18. Фотографія підпису студента.
19. Кнопка збереження фотографії підпису.
20. Інформація про авторів сайту.

Рисунок Г.9 – Структура графічного інтерфейсу вікна сторінки профілю

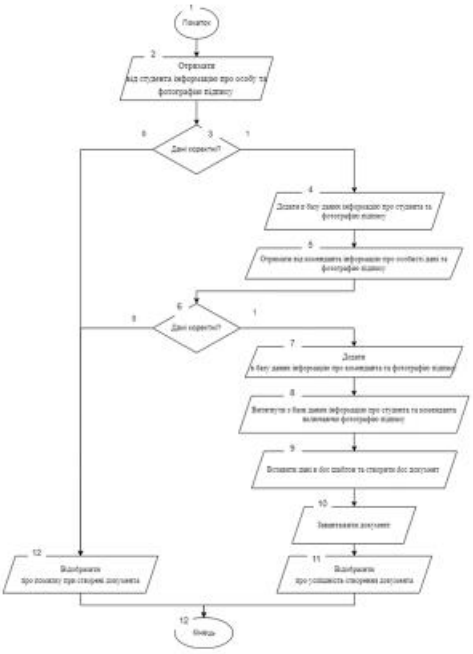


Рисунок Г.10 – Блок-схема алгоритму створення документів

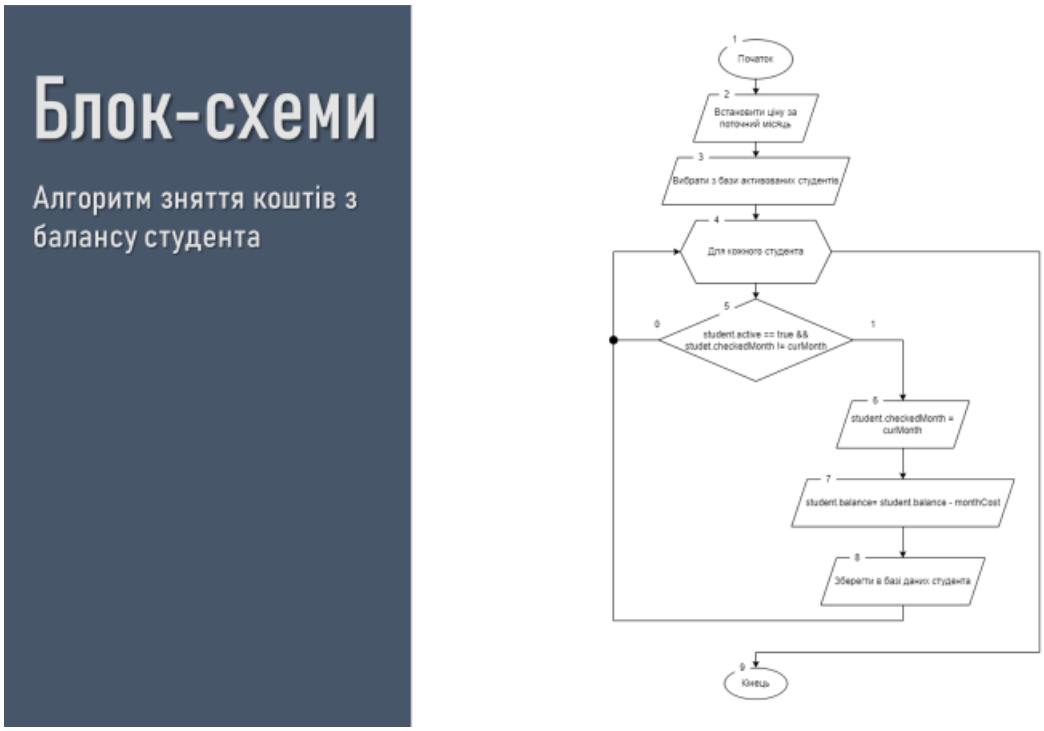


Рисунок Г.11 – Блок-схема алгоритму зняття коштів з балансу студента

Рисунок Г.12 – Блок-схема алгоритму нарахування коштів студенту та додавання квитанції в базу даних

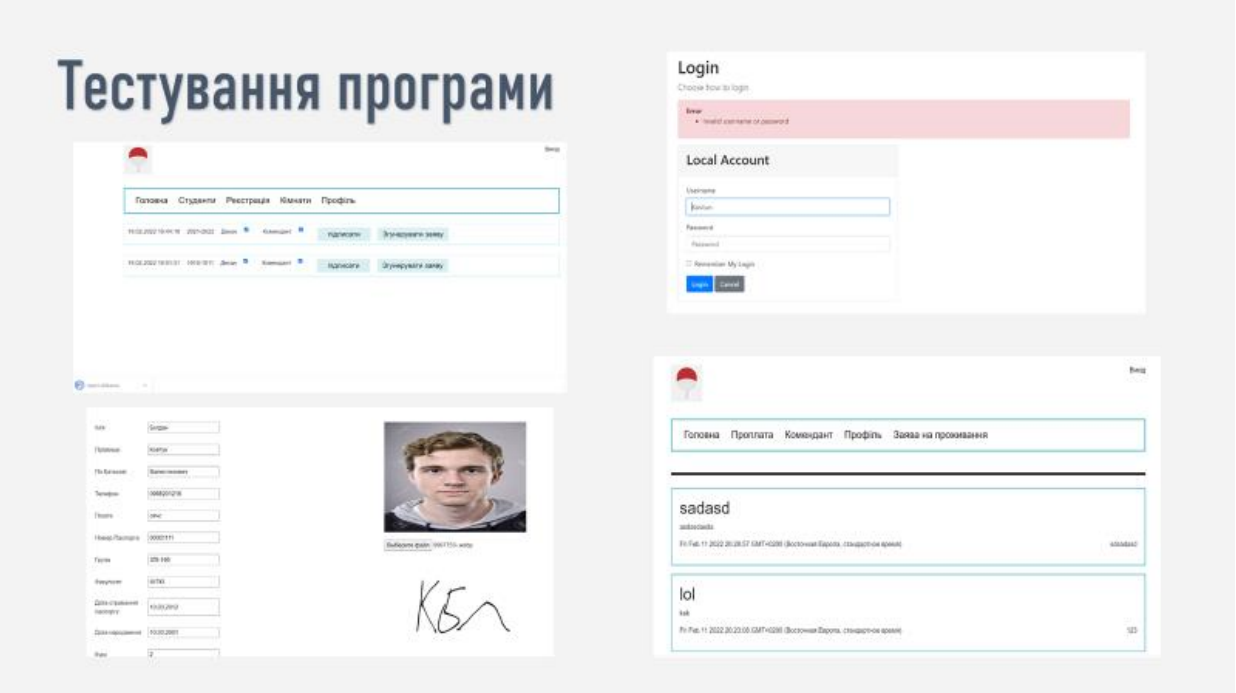


Рисунок Г.13 – Тестування програми

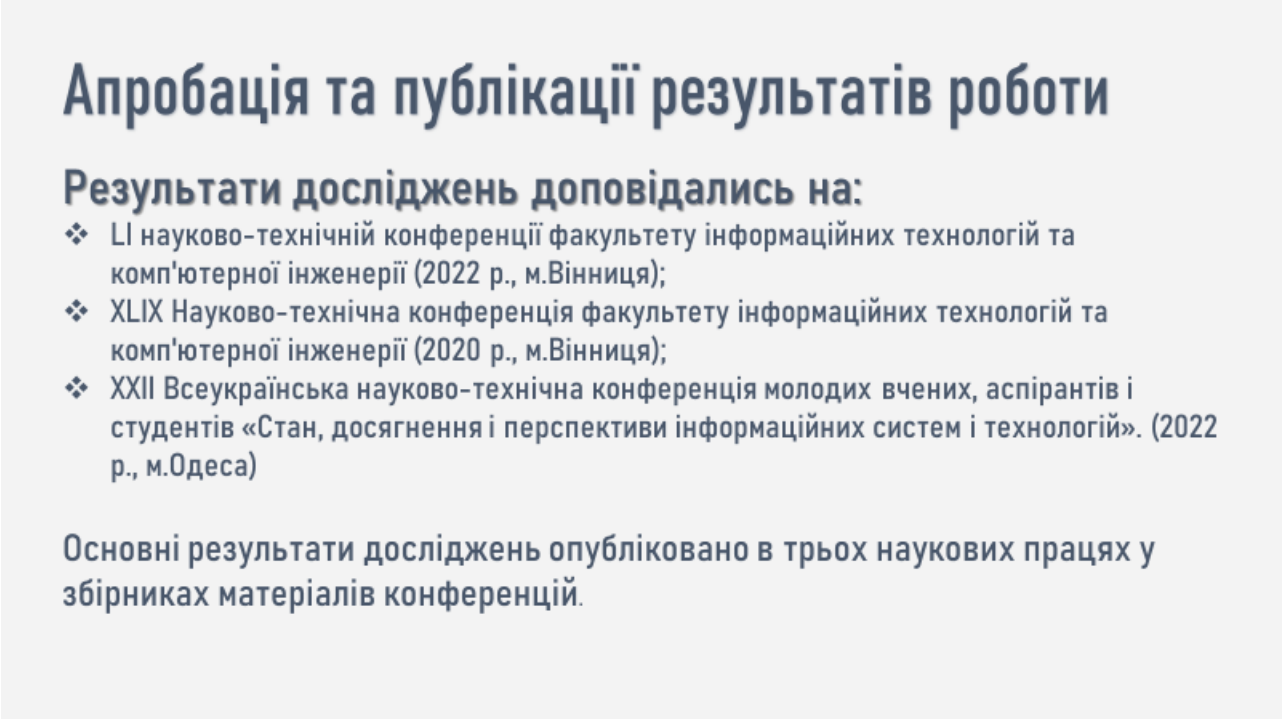


Рисунок Г.14 – Апробація та публікації результатів роботи

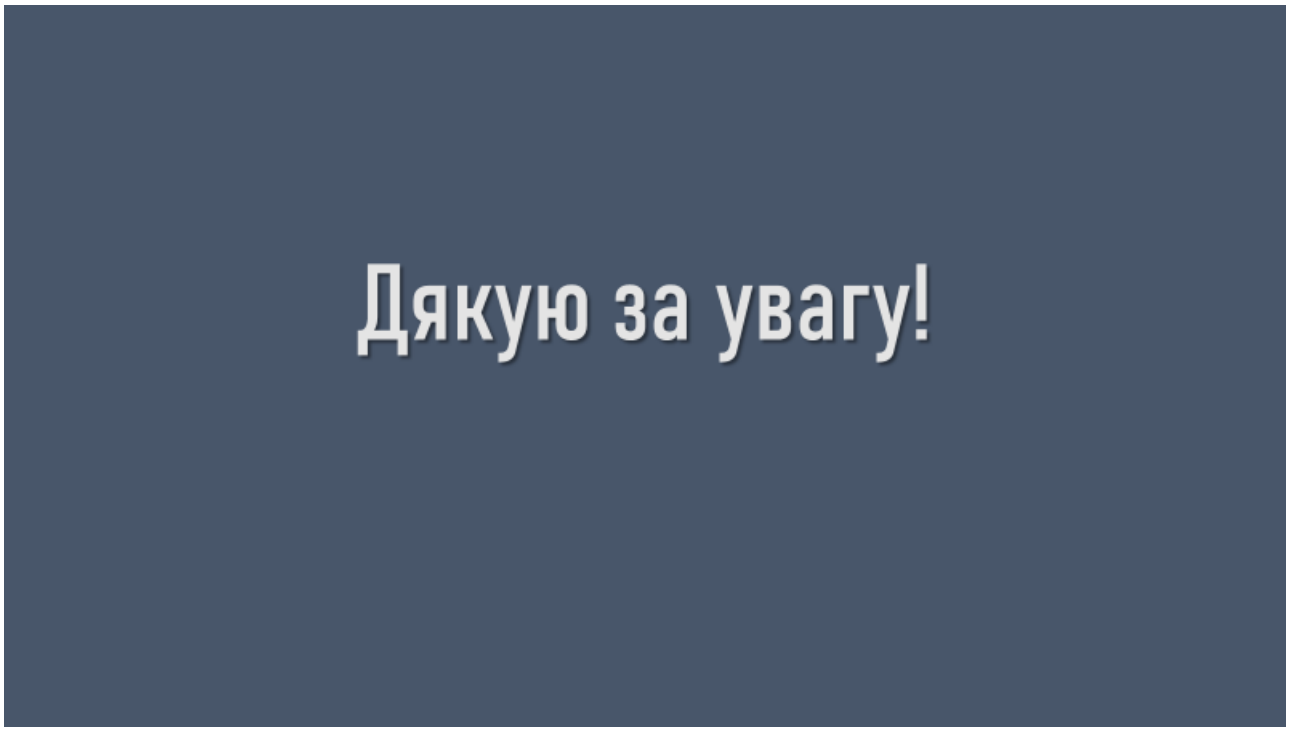


Рисунок Г.15 – Фінальний слайд