Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету»

Виконав: студент II курсу
групи 3ПІ-22м спеціальності
121 – Інженерія програмного забезпечення
Ковтун Богдан Валентинович

Керівник: к.т.н., доп. каф. ПЗ Романюк О.В.
«11» грудня 2023 р.

Опонент: к.т.н., доц. каф.ЗІ Куперштейн Л.М.
«11» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«11» грудня 2023 р.

ВНТУ – 2023

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«19» вересня 2023 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ковтуну Богдану Валентиновичу

1. Тема роботи – розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету.

Керівник роботи: Романюк Оксана Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від «18» вересня 2023 р. № 247.

2. Строк подання студентом роботи: 5 грудня 2023 р.

3. Вихідні дані до роботи: модель розробки – водоспадна; метод передачі повідомлень між сервісами – HTTP; система управління базами даних – MongoDB; вхідні дані – база даних з інформацією про студента, з інформацією про керуючий персонал, з підписами користувачів, з проплатами за проживання у гуртожитку завантажені студентами, з цінами за проживання за місяць у гуртожитку, з історією студентів проживання у гуртожитку; вихідні дані – розпізнана сума проплати та розрахований борг студента у вигляді текстової звіту, згенерована заява на проживання у гуртожитку у вигляді word документ середовище розробки – Visual Studio Code; мова програмування – TypeScript.

4. Зміст текстової частини: вступ; аналіз стану питання та постановка задач дослідження; розробка методів підвищення ефективності процесу управління гуртожитком університету та алгоритмів роботи програмного продукту; розробка програмного забезпечення для управління гуртожитком університету; тестування розробленого програмного продукту; економічна частина; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження; наукова новизна; практична цінність одержаних результатів; порівняльний аналіз аналогів;

формула обчислення боргу студента; метод і блок-схема алгоритму розпізн
суми проплати; метод і блок-схема алгоритму розрахунку боргу студента; м
і блок-схема алгоритму генерації документів; структура графічного інтерф
головного вікна застосунку для керуючого персоналу; структура графіч
інтерфейсу головного вікна застосунку для студента; тестування програм
забезпечення; експериментальні дослідження методу розпізнання суми проп
студента; економічна частина; апробація та публікації результатів роб
фінальний слайд.

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	19.09.2023	05.12.2023
5	Причепа І.В., к.е.н., доцент кафедри ЕПВМ	27.11.2023	28.11.2023

7. Дата видачі завдання 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Приміт
1	Аналіз стану питання та постановка задач дослідження	20.09. 2023 – 19.10.2023	вик
2	Розробка методів підвищення ефективності процесу управління гуртожитком університету	20.10.2023 – 24.10.2023	вик
3	Розробка алгоритмів роботи програмного застосунку	25.10.2023 – 19.11.2023	вик
4	Розробка програмного забезпечення	20.11.2023 – 24.11.2023	вик
5.	Тестування програмного застосунку та дослідження ефективності запропонованого методу	25.11.2023 – 26.11.2023	вик
6.	Економічна частина	27.11.2023 – 28.11.2023	вик
7.	Оформлення матеріалів до захисту МКР	29.11.2023 – 01.12.2023	вик

Студент (підпис) **Ковтун Б.В** (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи (підпис) **Романюк О.** (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.42

Ковтун Б. В. Розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету: магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. 108 с.

На укр. мові. Бібліогр.: 30 назв; рис.: 53; табл. 16.

У магістерській кваліфікаційній роботі було проведено детальний аналіз методів розпізнання проплати та розрахунку боргу студента.

Удосконалено метод розпізнання проплати студента, що базується на розпізнанні тексту зображення з використанням декількох стратегій для чеків з різних банків. Подальшого розвитку отримали метод динамічного розрахунку боргу студента та генерації документів з поточною інформацією. Розроблені методи підвищують ефективність процесу управління гуртожитку університету. Були проаналізовані аналоги застосунку, розроблено та перевірено роботу програмного забезпечення для підвищення ефективності процесу управління гуртожитку університету за допомогою таких засобів: мова програмування – C#, TypeScript; середовище розробки – Visual Studio 2022; фреймворки – .NET 6, NestJS, IdentityServer, React; архітектура – мікросервіси; база даних – MongoDB.

Отримані в магістерській кваліфікаційній роботі результати можна використати для підвищення ефективності процесу управління гуртожитку університету.

Ключові слова: проплата, розпізнання тексту, генерація документів, розрахунок боргу.

ABSTRACT

Kovtun B. V. Development of methods and software tools to increase the efficiency of the university dormitory management process. Vinnytsia: VNTU, 2023. 108 p.

In Ukrainian language. Bibliographer: 30 titles; fig. : 53; tabl. 16.

In the master's qualification work, a detailed analysis of the method of recognition of student overpayment was carried out, which increases the efficiency of the management process of the university dormitory.

The student payment recognition method based on image text recognition using several strategies for checks from different banks has been improved. The method of dynamic calculation of student debt and generation of documents with current information received further development. The developed methods increase the efficiency of the university dormitory management process. Analogues of the application were analyzed, the software was developed and tested to improve the efficiency of the university dormitory management process using the following tools: programming language – C#, TypeScript; development environment – Visual Studio 2022; frameworks – .NET 6, NestJS, IdentityServer, React; architecture – microservices; the database is MongoDB.

The results obtained in the master's qualification work can be used to improve the efficiency of the university dormitory management process.

Keywords: payment, text recognition, document generation, debt calculation.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сьогодні питання ефективності управління гуртожитком університету надзвичайно важливе, оскільки вища освіта стає все більш популярною серед молоді, і, таким чином, збільшується кількість студентів у гуртожитку. Збільшення проживаючих студентів у гуртожитку означає збільшення ресурсів, що потрібні для менеджменту гуртожитку і тим самим збільшення персоналу та витрат. Також існуючі рішення для менеджменту гуртожитку для студентів є неефективними або повністю неавтоматизованими. Існуючі програмні рішення не дозволяють генерувати документи для управління гуртожитком. Наявність додаткового програмного забезпечення для гуртожитку збільшить ефективність менеджменту та зменшить адміністративні витрати. Дозволить коменданту та іншому персоналу працювати дистанційно та отримувати всю необхідну інформацію за лічені секунди, що пришвидшить його роботу.

Взаємодія студента з гуртожитком повинна бути швидкою та зручною. Сучасні студенти широко використовують смартфони та різні цифрові сервіси. Вони віддали б перевагу заповненню форми в інтернеті, ніж походу до кабінету коменданта. Набагато зручніше отримувати інформацію про борг, реєстрацію проплати або необхідні документи зі всіма необхідними підписами з інтернету. Додаткове ПЗ зможе покращити взаємодію студента з гуртожитком [1].

Також важливим фактором є конкуренція серед університетів. Кожен університет конкурує за студентів і наявність ПЗ для менеджменту гуртожитку підвищує якість сервісу і тим самим конкурентоспроможність університету, привертаючи студентів якістю обслуговування та зручністю.

Додаткове ПЗ для студентів підвищить безпеку та контроль в гуртожитках університету [2]. ПЗ може допомогти університетам виявляти мешканців, які ігнорують встановлені правила, контролювати грошові надходження та забезпечувати прозорий фінансовий моніторинг боргу студента, що, в свою чергу, дозволить студенту дізнатись свій борг лише в декілька кліків, а не йти у

кабінет до коменданта, що зекономить студенту час для навчання.

Також на сьогоднішній день технологічний розвиток є надзвичайно великим. Існує багато сервісів статистики та різних державних цифрових сервісів. Додаткове ПЗ для гуртожитку може у майбутньому бути інтегроване з цими сервісами для ще кращого та більш ефективного менеджменту інформації користувача, що допоможе коменданту ефективніше керувати гуртожитком [3].

Тому актуальними є питання підвищення продуктивності та ефективності процесів управління гуртожитком для студентів, оскільки існуючі методи управління не задовольняють сучасні потреби. Це передбачає розробку нових методів і засобів підвищення ефективності процесів управління гуртожитком університету, а саме розробки додаткового ПЗ з функціоналом розпізнання суми проплати, менеджменту боргу студента та генерацією необхідних документів з коректною та новою інформацією.

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

Мета та завдання дослідження. Метою роботи є підвищення ефективності процесу управління гуртожитком університету шляхом розробки методів та програмних засобів для автоматизації процесів управління боргами, оплатами та інформацією про студента, а також генерації документів.

Основними задачами дослідження є:

– аналіз існуючих аналогів застосунку для управління гуртожитком університету, методів розпізнавання суми проплати по чеку та методів розрахунку боргу;
– розробка методу та алгоритму розпізнавання суми проплати з чеків різних банків, що завантажуються студентом;
– розробка методу та алгоритму автоматизації розрахунку боргу студента шляхом визначення місячної вартості оплати та фактичних проплат студента;
– розробка методу та алгоритму генерації документів шляхом отримання

інформації та підписів з профілів користувачів;

– розробка програмних компонент на основі запропонованих методів;
– експериментальні дослідження запропонованого методу розпізнавання суми проплати;
– тестування розроблених програмних компонент;
– розробити інструкцію користувача.

Об’єкт дослідження – процес автоматизованого управління гуртожитком університету.

Предмет дослідження – методи та засоби автоматизованого управління гуртожитком університету.

Методи дослідження. У процесі дослідження використовувались: лінійна алгебра з метою розробки методу та формули розрахунку боргу студента; теорія алгоритмів для розробки та удосконалення алгоритмів роботи програмного забезпечення; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

1. Удосконалено метод розпізнавання суми проплати, що завантажується студентом, у якому, на відміну від існуючих рішень, розпізнавання тексту відбувається з використанням декількох стратегій і за допомогою Regex, що дало можливість підвищити точність розпізнавання суми оплати з чеків різних банків на 26%.

2. Подальшого розвитку отримав метод автоматизації обліку боргу студента, що на відміну від відомих рішень, динамічно визначає борг шляхом встановлення комендантом помісячної вартості проживання, обліку місяців, що прожив студент у гуртожитку, та суми фактичних проплат студента, що дало можливість прискорити визначення боргу динамічно з появою нових даних.

3. Подальшого розвитку отримав метод генерації документів, який на відміну від інших рішень, генерує документи з релевантною інформацією, яка отримана з профілів користувачів, що дало можливість повністю автоматизувати процес створення документів.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та програмні засоби для підвищення ефективності процесу управління гуртожитком університету.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У наукових працях, опублікованих у співавторстві, автору належать такі результати: розпізнавання чеку проплати за допомогою технології OCR [4]; розробка методу розпізнання суми проплати з чеків різних банків [5]; аналіз методів розпізнання тексту на зображеннях [6].

Апробація матеріалів магістерської кваліфікаційної роботи. Результати роботи доповідалися на XVI міжнародній науково-практичній конференції «Інформаційні технології і автоматизація – 2023» (2023 р., Одеса), на міжнародній науково-практичній інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» (2023 р., Вінниця) та на LII науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКПВНТУ–2023) (2023 р., Вінниця).

Публікації. Основні результати досліджень опубліковано у трьох наукових публікаціях у збірниках матеріалів конференцій.

1 АНАЛІЗ СТАНУ ПИТАННЯ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану питання

Гуртожиток для студента є надзвичайно важливим етапом життя студента. Гуртожиток є невід'ємною складовою студентського життя, де молодь спілкується, навчається і росте як особистість. Це місце, де студенти з різних куточків країни та світу зустрічаються, вивчають культурні особливості один одного, діляться досвідом і знаннями. Гуртожиток стає платформою для отримання важливих знань та вмінь, що будуть потрібні у подальшому житті студента. Гуртожиток створює сприятливе середовище для навчання та саморозвитку. Студенти мають зручний доступ до бібліотеки, комп'ютерних лабораторій та інших навчальних ресурсів, що сприяє поглибленню знань та вдосконаленню навичок. Наявність однодумців та стимулюючого середовища підтримує академічний прогрес та розвиток. Якість взаємодії студента з гуртожитком та ефективність управління гуртожитком впливає на навчання студента та на його здоров’я, оскільки навчання в університеті є надзвичайно стресовим. Ефективне управління гуртожитком дозволить концентруватися студенту на навчанні, а не на вирішенню проблем з проживанням у гуртожитку.

На сьогоднішній день управління гуртожитками університетів зазвичай базується на традиційних методах та процедурах. Основні проблеми, ідентифіковані у сфері управління гуртожитками, включають:

Ручне управління – велика кількість процесів, такі як розподіл кімнат та облік платежів, виконуються вручну. Це є затратно по часу, оскільки щоб дізнатися потрібну інформацію про студента коменданту потрібно перебрати велику кількість документів. Також це може призвести до помилок. Виконання ручної роботи має в собі більше ризиків зробити помилку, оскільки потрібно тримати багато інформації у голові. Ручне управління швидко приводить до втоми і тому впливає на ефективність управління гуртожитком. Також мінусом ручного управління є те, що сервіс не доступний цілодобово, а визначається лише робочим днем персоналу. Автоматизація дозволить певну частину

функціоналу зробити доступною у будь яку пору доби.

Безпека і контроль – недостатність системи контролю та безпеки може призводити до небажаних подій та порушень в гуртожитках. Відсутність системи перевірки проплат на предмет достовірності може нести великі проблеми для гуртожитку та для студента. Комендант при перевірці проплати може не побачити, що рахунок отримувача не співпадає з рахунком гуртожитку, або студент може надати однакові квитанції з ознієї транзакції і видати ці квитанції як дві транзакції. Тому потрібно мати валідатор проплати студентів за гуртожиток на етапі надання проплати гуртожитку, щоб не створювати проблем гуртожитку, а також самим студентам.

Зручність взаємодії студента з гуртожитком – недостатня якість взаємодії студента з гуртожитком, оскільки управління гуртожитком відбувається вручну майже всюди, то студенту щоб взаємодіяти з гуртожитком потрібно йти в кабінет до коменданта або до університету [7]. Зазвичай студенти намагаються вирішити проблеми з гуртожитком в останній день і це призводить до утворення величезних черг і студентам потрібно стояти в черзі по декілька годин щоб отримати можливість зустрітись з комендантом. Також щоб дізнатися про поточний борг потрібно також йти до коменданта. Відсутність можливості хоча б частково автоматизувати процес управління та перенести сервіси обслуговування в інтернет надзвичайно сильно погіршують зручність взаємодії студента з гуртожитком та ефективність його управління [8].

Для вирішення зазначених проблем потрібно розробити власне програмне забезпечення, оскільки аналоги не забезпечують весь функціонал управління гуртожитком. Розробка власної системи завантаження проплат студентів вирішить проблему перевірки правильності внесених проплат студентом, що покращить ефективність управління гуртожитком та збільшить безпеку та контроль. Розробка власної системи розрахунку боргу для студента повністю автоматизує процес обчислення боргу студента, що підвищить ефективність взаємодії студента з гуртожитком та покращить процес управління. Розробка власної системи генерації документів дозволить студентам та управляючому

персоналу гуртожитку генерувати документи зі всією необхідною інформацією, що сильно підвищує зручність взаємодії студента з гуртожитком. Взаємодія з програмним забезпеченням буде відбуватися за допомогою браузера та інтернету. Програмне забезпечення для гуртожитку буде мати web-інтерфейс, що є зручним та зрозумілим для користувачів, оскільки він є знайомим для користувачів. Хостинг програмного продукту буде відбуватися за допомогою Microsoft Azure, що дозволить користувачам мати доступ до сайту з будь якої частини світу.

1.2 Порівняльний аналіз аналогів

Головною метою розробки є підвищення ефективності процесу управління гуртожитком університету за рахунок розробки нових методів розпізнання проплати, що завантажується студентом, обліку боргу студента та генерації документів, які будуть реалізовані у веб-застосунку. На даний момент існує не така велика кількість програмного забезпечення для підвищення ефективності процесу управління гуртожитком.

Розглянемо такі аналоги:

– Creatrix Campus;

– StarRez;

– Ellucian Hosing;

– Entrata Student.

Creatrix Campus – це програмне рішення для управління студентськими гуртожитками та житловими питаннями в учбових установах. Воно розроблене для полегшення процесів, пов'язаних зі студентським житлом, такими як бронювання, облік мешканців, оплата, обслуговування та звітність. Перевагами програмного продукту є наявність інструментарію для ефективного розподілення кімнат та місць у гуртожитку, можливість бронювати кімнати. Creatrix Campus може забезпечувати звітність та аналітику для університетів, що допомагає в розумінні тенденцій та удосконаленні управління. Також присутній менеджмент боргу студента та його проплати. До мінусів можна віднести

вартість, оскільки провадження та підтримка такого програмного забезпечення може бути дорогим для університетів та коледжів. Також важливим мінусом є складність системи. Впровадження нової системи може вимагати часу та зусиль для навчання персоналу та студентів її використанню. Компанією розробником є Anubavam. Користувацький інтерфейс Creatrix Campus наведено на рис. 1.1.

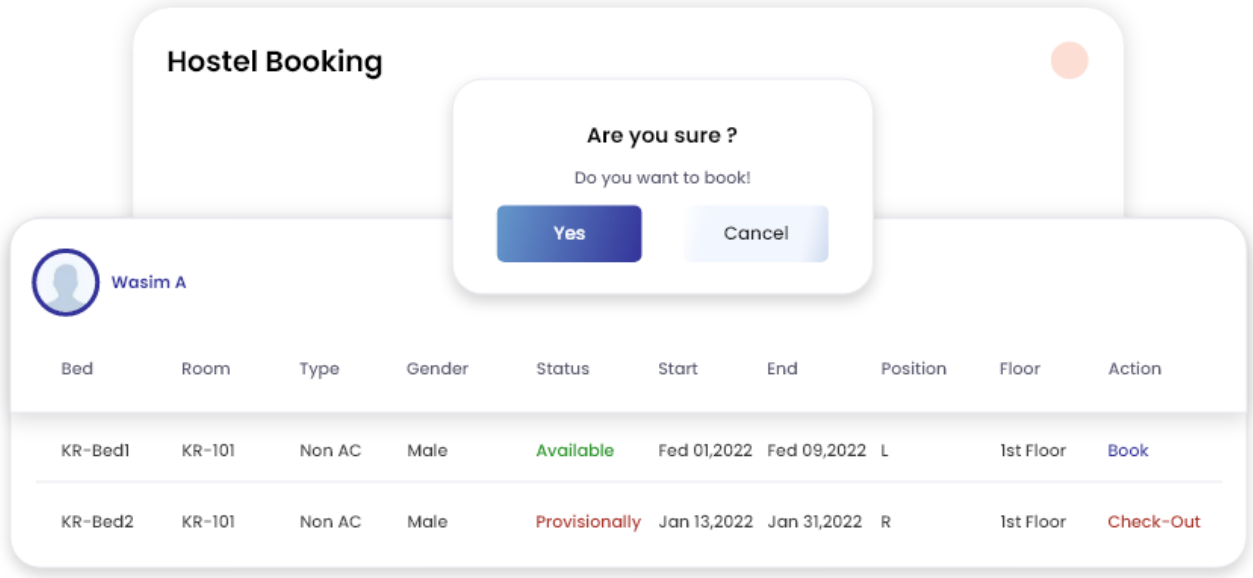


Рисунок 1.1 – Користувацький інтерфейс застосунку Creatrix Campus

StarRez – це програмне забезпечення для управління студентськими гуртожитками, яке широко використовується в освітній сфері. Ця система спрощує процеси реєстрації студентів, контролю доступу до гуртожитків, управління резерваціями та облік оплати за житло. Розроблене StarRez програмне забезпечення допомагає університетам та коледжам забезпечити комфортні та безпечні житлові умови для своїх студентів. Плюси StarRez включають універсальність – відповідає різним видам освітніх закладів, зручний інтерфейс для користувачів і адміністраторів, можливість проведення аналізу даних та створення звітів для прийняття рішень, інтеграцію з іншими системами освітніх закладів. Також StarRez має мінуси. Вартість може бути досить високою, що може бути задорого для невеликих освітніх закладів. Також потрібне

навчання та підтримка для користувачів і адміністраторів для ефективного використання системи. Для деяких невеликих гуртожитків ця система може бути занадто складною. Крім того, StarRez потребує постійного доступу до Інтернету, оскільки це хмарне рішення. Розробником StarRez є компанія StarRez. Користувацький інтерфейс StarRez наведено на рисунку 1.2.

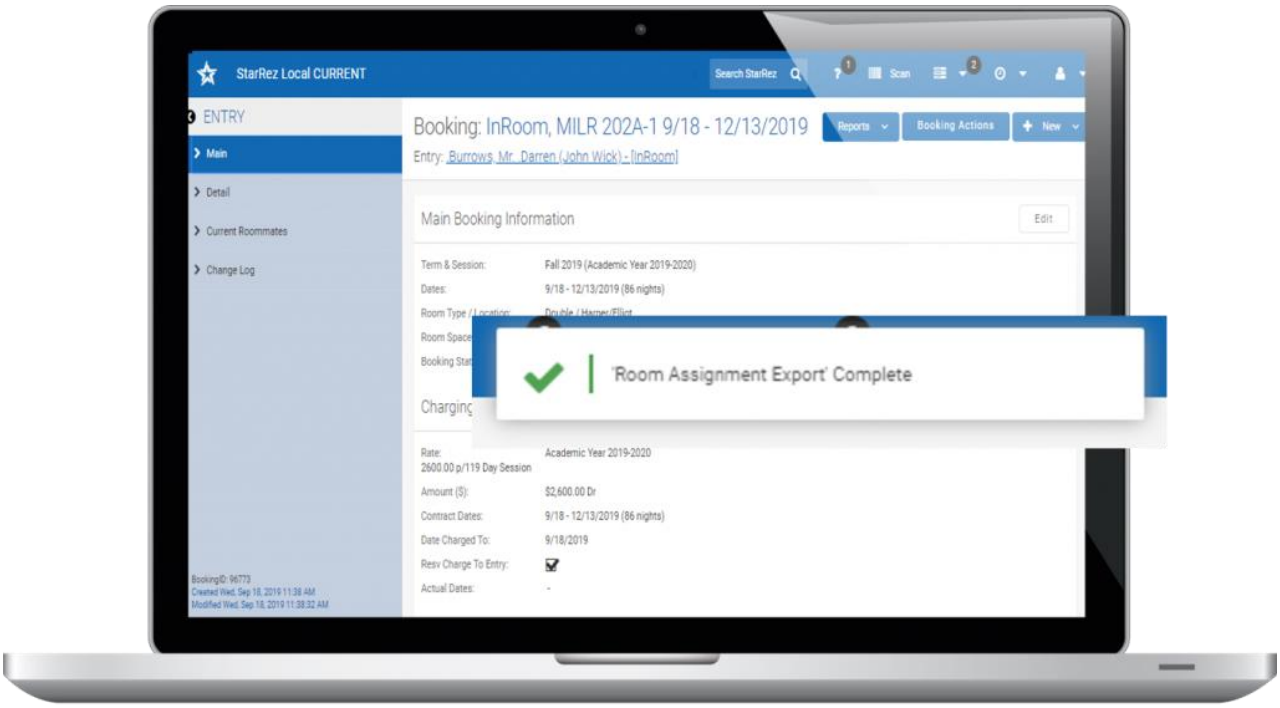


Рисунок 1.2 – Користувацький інтерфейс застосунку StarRez

Ellucian Housing – це програмне забезпечення для управління студентськими гуртожитками та житловими умовами, розроблене компанією Ellucian. Ця система призначена для вищих навчальних закладів і допомагає університетам і коледжам управляти гуртожитками та забезпечувати комфортні умови проживання для студентів. Плюсами Ellucian Housing є універсальність, система може бути використана різними видами освітніх закладів, включаючи університети, коледжі та інші. Також Ellucian Housing пропонує зручний інтерфейс як для користувачів, так і для адміністраторів, що полегшує роботу з системою. Ellucian Housing надає інструменти для аналізу даних та створення звітів, що сприяє прийняттю правильних рішень. Також до мінусів можна

віднести велику ціну за обслуговування системи та її впровадження, можливість перенавантаження, та потрібність інтернету для функціонування. Користувацький інтерфейс Ellucian Housing наведено на рисунку 1.3.

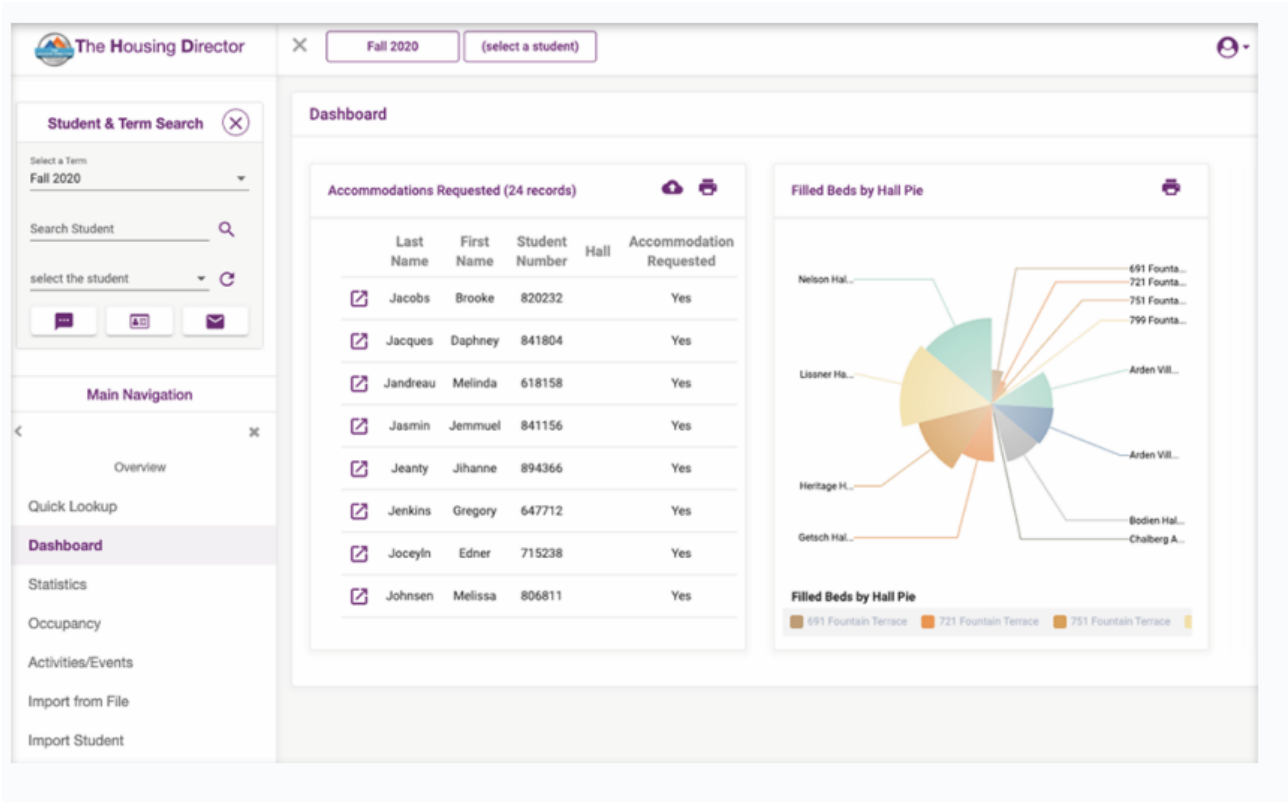


Рисунок 1.3 – Користувацький інтерфейс застосунку Ellucian Housing

Entrata Student – це програмне забезпечення, розроблене компанією Entrata, яке призначене для управління студентськими гуртожитками та житловими умовами в освітніх закладах. Ця система спрощує реєстрацію студентів, контроль доступу до гуртожитків, оплату за житло та інші аспекти житлового обліку для вищих навчальних закладів. Позитивним аспектом програмного забезпечення є зручний інтерфейс, можливість мати аналітику та звітність, можливість інтегруватися з різними освітніми системами. Головним мінусом Entrata Student є велика вартість впровадження та обслуговування. Також до мінусів можна віднести кольори інтерфейсу застосунку, що є яскравими і втомлюють користувача. Користувацький інтерфейс Entrata Student наведено на рисунку 1.4.

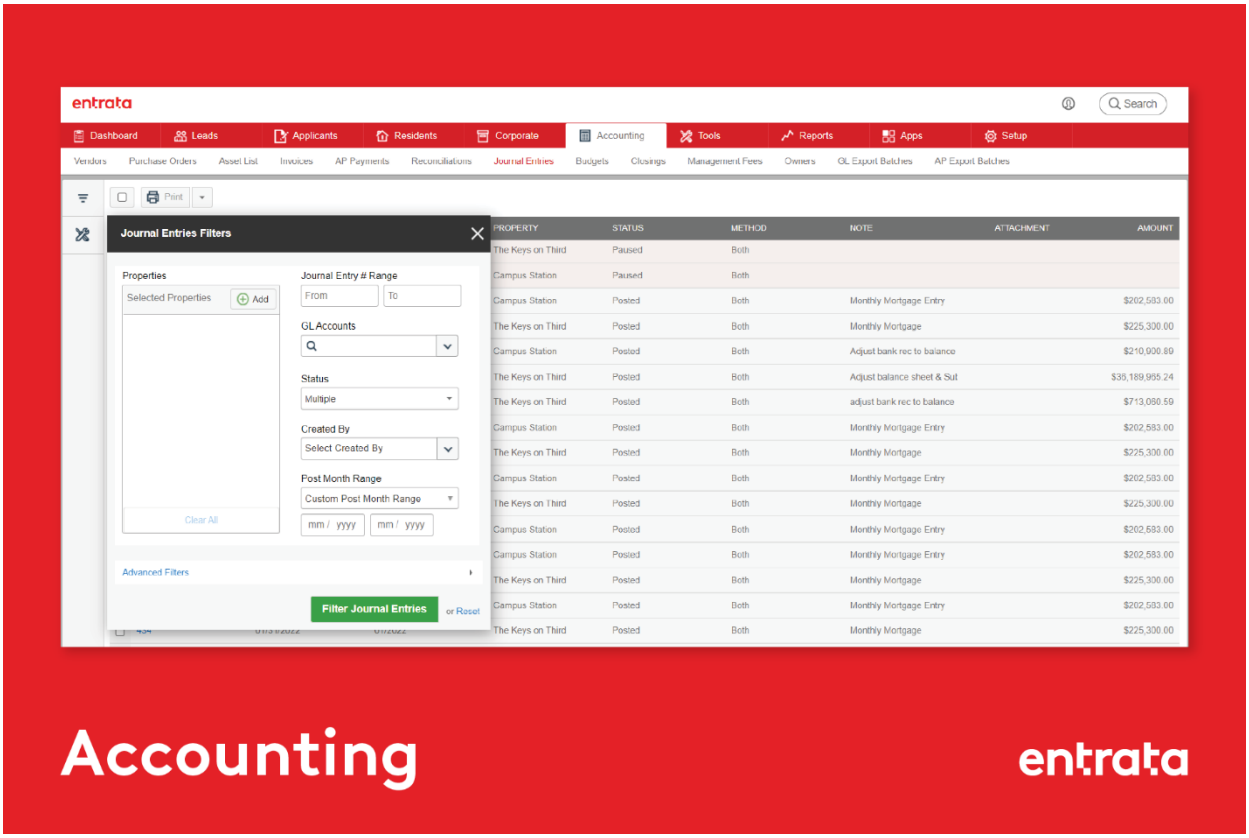


Рисунок 1.4 – Користувацький інтерфейс застосунку Entrata Student

Створимо таблицю з результатами аналізу усіх аналогів розроблюваного програмного продукту, визначимо головні переваги та недоліки їх функцій покращення ефективності процесу управління гуртожитку для університету та зрівняємо із розроблюваним веб-застосунком під назвою “Dormament Master”. Результат порівняння наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльні характеристики програмних продуктів

Критерій	Creatrix Campus	StarRez	Ellucian Hosing	Entrata Student	Dormament Master
Можливість валідації проплати	0	0	0	0	1

Продовження таблиці 1.1 – Порівняльні характеристики програмних продуктів

Можливість менеджменту боргу	1	1	0	1	1
Можливість генерації документів для поселення	0	1	1	1	1
Менеджмент інформації про студента	1	1	1	1	1
Підсумковий результат	2	3	2	3	4

На основі результатів порівняння функцій покращення ефективності процесу управління гуртожитком для університету аналогів із розроблюваним програмним застосунком «Dormament Master» було зроблено висновок, що створення власного застосунку є доцільним. В результаті розробки отримаємо програмний продукт, що вирішує недоліки існуючих аналогів та збільшує функціонал для збільшення ефективності процесу управління гуртожитком.

1.3 Аналіз методів розпізнання суми проплати з чеків, що завантажуються студентом

Розпізнавання суми проплати, що завантажується студентом за допомогою OCR є однією з найважливіших функцій для застосунку, оскільки це дозволяє збільшити безпеку для гуртожитку. Є різні алгоритми та бібліотеки, що імплементують OCR.

OCR (Optical Character Recognition) або розпізнавання оптичного тексту – це технологія, яка дозволяє комп'ютерам автоматично розпізнавати та виділяти текст зі зображень, фотографій або сканів документів [9]. OCR дозволяє перетворювати текст, зображений на фізичних носіях, у машинночитабельний текст, який можна редагувати, зберігати, шукати та аналізувати на комп'ютері. Ця технологія використовується в різних галузях, включаючи обробку документів, розпізнавання тексту на фотографіях, автоматизовану обробку проплат та багато інших застосувань.

Існує кілька методів і підходів до реалізації OCR-систем. Ось деякі з них:

1. Методи на основі шаблонів (Template-based OCR). В цьому методі створюються шаблони для кожного символу або слова, і текст розпізнається шляхом порівняння зі шаблонами;

2. Методи на основі структури (Structural OCR). Вони використовують аналіз структури тексту, зокрема розташування символів і їхні відносні позиції на зображенні. Це може включати в себе визначення ліній, колонок, таблиць тощо;

3. Методи на основі нейронних мереж (Neural Network OCR). Використовуються глибокі нейронні мережі, такі як згорткові нейронні мережі (CNN) або рекурентні нейронні мережі (RNN), для автоматичного визначення тексту на зображенні [10];

4. Методи на основі візуальних ознак (Feature-based OCR). Вони використовують візуальні ознаки, такі як форми та структури символів, текстури, контраст, для розпізнавання тексту.

Очевидним і найкращим рішенням є вибір методу на основі нейронних мереж, оскільки це дозволяє розпізнавати текст точно для різних видів шрифтів та для різної величини символів. Також доцільно використовувати існуючу бібліотеку для OCR Tesseract з використанням алгоритму Long Short-Term Memory, що використовує нейронні мережі. Бібліотека Tesseract є безкоштовною, підтримує мову програмування C# , що робить цю бібліотеку доцільним вибором при розробці застосунку для підвищення ефективності процесу управління гуртожитку університету.

При використанні нейронних мереж текст на картинці розпізнається та передається в змінну, зазвичай цей текст перемішаний і не завжди можливо віднайти потрібні дані у цьому тексті. Тому найкращим рішенням є розпізнавати певні ділянки на картинці, а потім за допомогою Regex отримувати інформацію, що потрібна.

Отже, для розпізнання проплати, що завантажується студентом, доцільно використовувати нейронні мережі, а саме бібліотеку Tesseract. Розпізнання в

картинці доцільно виконувати по певній ділянці і отримувати необхідну інформацію за допомогою Regex.

1.4 Аналіз методів розрахунку боргу студента

Розрахунок боргу студента є одним з найважливіших компонентів застосунку для підвищення ефективності процесів управління гуртожитком, оскільки дану послугу можна зробити онлайн. Різні гуртожитки університету використовують різні методи нарахування боргу, а саме:

1. Фіксована плата за місяць. Цей метод передбачає встановлення фіксованої щомісячної або семестрової плати за проживання в гуртожитку. Студентам потрібно сплачувати однакову суму грошей незалежно від того, скільки часу вони проводять у гуртожитку. Цей метод досить простий і передбачуваний;

2. Плата за день/тиждень: Деякі університети встановлюють ціну за день, тиждень або навіть за годину проживання в гуртожитку. Студенти сплачують лише за той період, коли вони фактично користуються гуртожитком. Цей метод може бути більш справедливим для студентів, які користуються гуртожитком лише частково;

3. Система бод-системи (Pay-as-you-go): У цій системі студенти сплачують лише за ті послуги та зручності, які вони фактично використовують. Наприклад, вони можуть платити окремо за інтернет, прибирання, пральню, тощо;

4. Плата за типом кімнати: В інших випадках вартість проживання може визначатися на основі типу кімнати, наприклад, одномісна, двомісна або кімната "люкс". Студенти, які обирають більш комфортні кімнати, сплачують більше.

Найпоширенішою в Україні системою нарахування боргу студенту є фіксована плата за місяць, тому доцільно вибрати саме цей метод нарахування боргу. Для реалізації цього методу потрібно встановити для кожного місяця ціну проживання. І сума боргу для студента буде визначатися сумую всіх цін проживання за місяць, що прожив студент мінус проплати студента.

Доцільно використовувати лише підтверджені проплати студента, оскільки розроблюваний веб-застосунок використовує OCR як розпізнання суми на проплаті і завжди є якийсь відсоток, що буде якась помилка, тому потрібно додати можливість мануального вводу ціни для проплати та його підтвердження керуючим персоналом гуртожитку. У результаті борг студента визначається за формулою:

$$\text{debt} = \sum_{i=1}^{m} \text{month\_price} - \sum_{i=1}^{p} \text{approved\_payment},$$

де debt – це борг студента за проживання у гуртожитку, month_price – ціна проживання за місяць, approved_payment – проплата студента, що була підтверджена керуючим персоналом, m – кількість місяців, p – кількість проплат.

Отже, у веб-застосунку для підвищення ефективності процесів управління гуртожитку для компоненту визначення боргу доцільно використовувати фіксовану ціну проживання за місяць як метод нарахування боргу і визначати борг як різницю сум всіх цін місяців та сум всіх підтверджених проплат студента. Підтверджувати проплату з чеку, що завантажується студентом, буде керуючий персонал шляхом мануального перегляду та підтвердження.

1.5 Постановка задач розробки

Після проведення аналізу стану питання щодо підвищення ефективності процесу управління гуртожитком університету, було визначено наступні завдання, що потрібно виконати для розробки відповідного програмного забезпечення:

1. Розробити метод та алгоритм розпізнавання суми проплати з чеків різних банків, що завантажуються студентом;
2. Розробити метод автоматизації та алгоритм розрахунку боргу студента шляхом визначення місячної вартості оплати та фактичних проплат студента;
3. Розробити метод та алгоритм генерації документів шляхом отримання інформації та підписів з профілів користувачів;

4. Розробити програмні компоненти на основі запропонованих методів;
5. Провести експериментальні дослідження запропонованого методу розпізнавання суми проплати;
6. Провести тестування розроблених програмних компонент;
7. Розробити інструкцію користувача.

Технічне завдання наведено в додатку А.

1.6 Висновки

У першому розділі було проведено аналіз стану питання програмного забезпечення для підвищення ефективності процесу управління гуртожитком для університету. Була досліджена предметна область та було проведено порівняльний аналіз аналогів з розроблюваним ПЗ, за результатами порівняння, було визначено доцільність розробки власного програмного продукту для підвищення ефективності процесу управління гуртожитку для університету, оскільки розроблюваний програмний продукт не містить недоліків аналогів і також повністю забезпечує потрібний функціонал.

Також було проведено аналіз методів розпізнання суми проплати з чеків, що завантажуються студентом. У результаті було визначено, що доцільно використовувати нейронні мережі для розпізнання тексту зображення, а саме бібліотеку Tesseract. Також для збільшення якості розпізнання тексту зображення потрібно розпізнавати тільки певну ділянку зображення. А для знаходження потрібної інформації використовувати Regex.

Було проведено аналіз методів розрахунку боргу студента. Було вирішено, що фіксована ціна за місяць буде використана як метод нарахування боргу. Тобто студент буде сплачувати за місяці, що прожив у гуртожитку. Також було вирішено, що доцільно розраховувати борг студента як різницю сум всіх прожитих місяців студентом у гуртожитку та підтверджених проплат студента. Також було проведено постановку задач розробки.

2 РОЗРОБКА МЕТОДІВ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПРОЦЕСУ УПРАВЛІННЯ ГУРТОЖИТКОМ УНІВЕРСИТЕТУ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

2.1 Розробка методу та алгоритму розпізнання суми проплати з чеків, що завантажуються студентом

Основною проблемою при розпізнанні тексту є те, що розпізнаний текст не формує чітку структуру, текст може буде перемішаний, тобто наприклад якщо текст на картинці є справа та зліва, то можливий варіант того, що текст буде вважатися одним рядком, хоча це не так, тобто текст може бути перемішаний і не формувати логічну інформацію. У такому випадку потрібну інформацію з перемішаного тексту буде витягнути проблематично, якщо потрібна інформація не буде мати якогось унікального паттерну.

Вирішенням даної проблеми є розпізнання тексту лише в певній ділянці картинки, де потрібна інформація буде мати унікальний паттерн. Знаходити потрібну інформацію можна з використанням інструменту Regex.

Також студент може завантажувати різні картинки проплат, що можуть мати різну структуру, оскільки це можуть бути проплати різних банків. При розпізнанні тексту кожен вид проплати банку буде мати свою унікальну структуру і розпізнання суми проплати буде проблематичним. Для вирішення цієї проблеми можна задіяти паттерн стратегії та для кожної проплати банку розробити свою імплементацію стратегії. І коли студент завантажить проплату, система буде розпізнавати текст для кожного банку і вибирати ту стратегію в якій було знайдено найбільше потрібної інформації.

Отже, метод розпізнання суми проплати, що завантажується студентом буде перевіряти лише виділену область та шукати потрібну інформацію за унікальним паттерном, використовуючи Regex. Також для вирішення проблем розпізнання суми проплати для різних банків було вирішено використовувати паттерн стратегії і для кожної проплати банку реалізувати свою стратегію розпізнання і вибирати ту стратегію, де було розпізнано найбільше потрібної

інформації.

Блок-схема даного алгоритму наведена на рисунку 2.1.

Крок 1. Початок.

Крок 2. Завантажити проплату та зберегти це значення у змінній image.

Крок 3. Присвоїти змінній maxRecognitions значення -1, а paymentInfoResult присвоїти null.

Крок 4. Запустити цикл, який буде перебирати всі наявні стратегії розпізнання paymentCheckStrategies. Якщо стратегії закінчилися – перейти до кроку 18, якщо ні – до кроку 5.

Крок 5. Присвоїти змінній recognizeResult значення 0 та присвоїти змінній engine об’єкт бібліотеку TesseractEngine.

Крок 6. Зберегти в змінну destinationCardNumber значення функції розпізнання GetDestinationCardNumber для стратегії в змінній recognitionStrategy, функція розпізнає номер рахунку отримувача грошей.

Крок 7. Зберегти в змінну checkPaymentId значення функції розпізнання GetCheckPaymentId для стратегії в змінній recognitionStrategy, функція розпізнає унікальний номер транзакції.

Крок 8. Зберегти в змінну checkPaymentPrice значення функції розпізнання GetCheckPaymentPrice для стратегії в змінній recognitionStrategy, функція розпізнає суму проплати.

Крок 9. Створити новий об’єкт checkPaymentInfo з розпізнаною інформацією.

Крок 10. Здійснити перевірку чи destinationCardNumber не дорювнює null, якщо так – перейти до кроку 11, якщо ні – до кроку 12.

Крок 11. Збільшити змінну recognizeResult на 1.

Крок 12. Здійснити перевірку чи checkPaymentId не дорювнює null, якщо так – перейти до кроку 13, якщо ні – до кроку 14.

Крок 13. Збільшити змінну recognizeResult на 1.

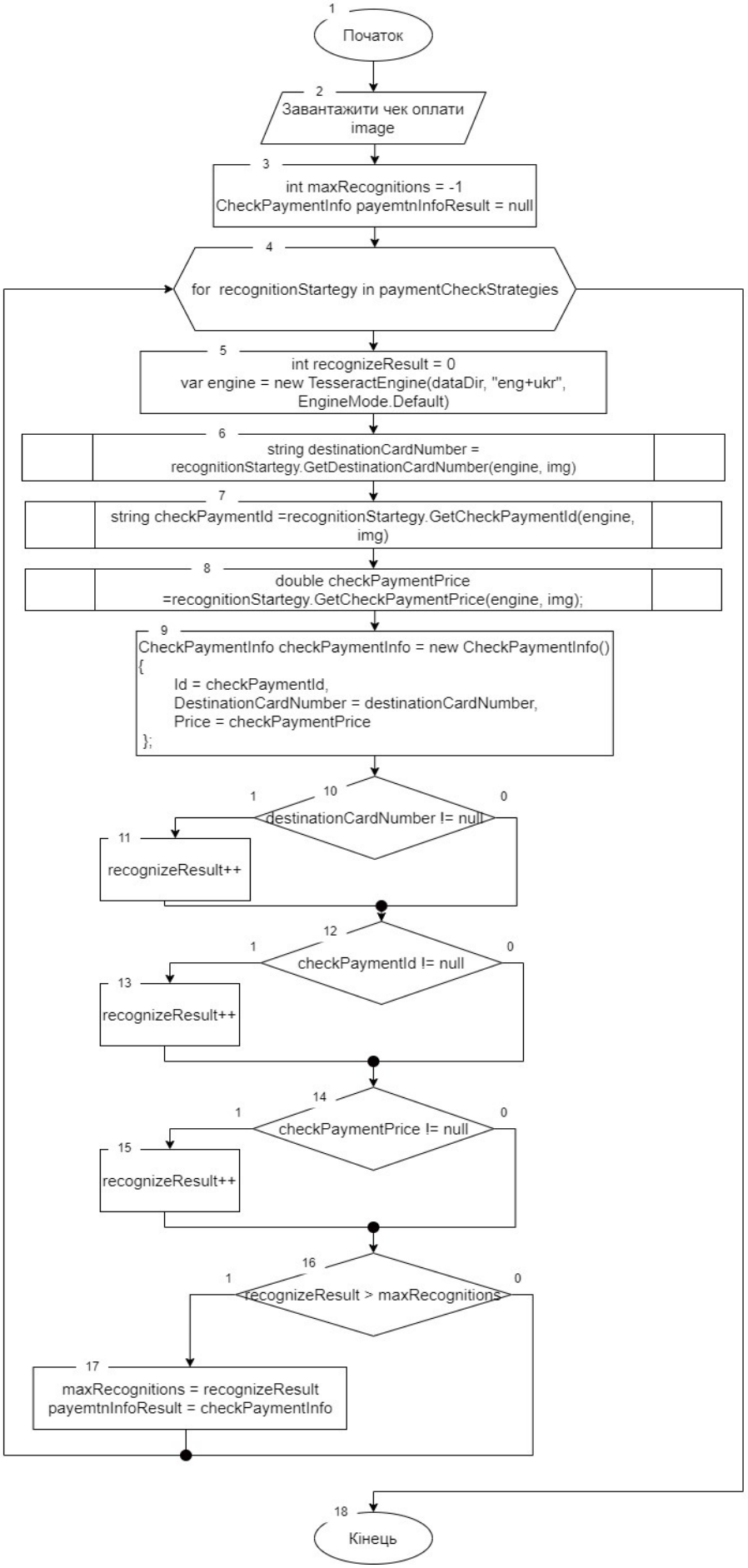


Рисунок 2.1 – Блок-схема алгоритму розпізнання суми проплати

Крок 14. Здійснити перевірку чи checkPaymentPrice не дорювнює null, якщо так – перейти до кроку 15, якщо ні – до кроку 16.

Крок 15. Збільшити змінну recognizeResult на 1.

Крок 16. Здійснити перевірку чи recognizeResult більше за maxRecognitions, якщо так – перейти до кроку 17, якщо ні – до кроку 4.

Крок 17. Присвоїти змінній maxRecognitions значення recognizeResult, а змінній paymentInfoResult значення checkPaymentInfo.

Крок 18. Кінець.

Таким чином відбувається розпізнання суми проплати з чеків, що завантажуються студентом.

2.2 Розробка методу та алгоритму розрахунку боргу студента

Розрахунок боргу студента є однією з найважливіших функціональностей розроблюваного застосунку, оскільки це допоможе автоматизувати процес нарахування боргу студента та менеджменту боргу. Для реалізації моніторингу боргу студента за проживання у гуртожитку університету за місяць потрібно встановити ціну проживання за кожен місяць.

Також потрібно розробити механізм для нарахування прожитих місяців у гуртожитку для студента. Найочевиднішим вибором є зберігання всіх прожитих місяців у таблицю або колекцію. Такий підхід є зрозумілим але складним в реалізації, оскільки потрібно буде розробляти додатковий інструмент, що буде перевіряти всіх студентів університету чи вони проживають у поточний місяць, програма буде викликатися кожного місяця. Кращим рішенням буде створити та зберігати в базі даних об’єкт rentRecord, що має у собі місяць та рік початку проживання у гуртожитку та місяць та рік закінчення проживання у гуртожитку. Такий підхід не вимагає створення додаткового інструментарію та дозволяє ефективно дізнаватись які місяці прожив студент у гуртожитку.

Також при розрахунку боргу студента потрібно брати до уваги проплати студента. Оскільки при розпізнанні суми проплати можуть виникати помилки, то доцільно надавати можливість керуючому персоналу перевіряти та змінювати

суму проплати, тобто при зміні суми проплати, при розрахунку боргу буде використовуватись сума введена керуючим персоналом, а не розпізнана системою сума.

Борг студента буде визначатися за формулою:

$$\text{debt} = \sum_{i=1}^{m} \text{month\_price} - \sum_{i=1}^{p} \text{approved\_payment},$$

де debt – борг у гривнях, можливе мінусове число, якщо студент вніс більшу суму ніж сума місяців, що прожив студент у гуртожитку; month_price – ціна за місяць; m – кількість місяців, що прожив студент у гуртожитку; approved_payment – перевірена керуючим персоналом проплата; p – кількість перевірених проплат студента.

Блок-схема алгоритму розрахунку боргу студента наведена на рисунку 2.2.

Крок 1. Початок.

Крок 2. Отримати від користувача id юзера та записати значення в userId.

Крок 3. Отримати з бази даних історії проживання у гуртожитку та записати значення у rentHistories.

Крок 4. Записати в змінну rentHistoryDictionary значення пустого об’єкту.

Крок 5. Запустити цикл, який буде перебирати елементи списку історій проживання у гуртожитку (rentHistories). Якщо список закінчився – перейти до кроку 11, якщо ні – до кроку 6.

Крок 6. Записати значення початкової дати для історії проживання у гуртожитку у змінну currentDate. Також записати у змінну endDate поточну дату.

Крок 7. Здійснити перевірку, чи змінна rentHistory.isActive (чи поточна історія проживання активна) має значення false. Якщо так, то здійснюється перехід до кроку 8, якщо ні – до кроку 9.

Крок 8. Записати значення rentHistory.endRentDate (кінцева дата історії проживання у гуртожитку) у змінну endDate.

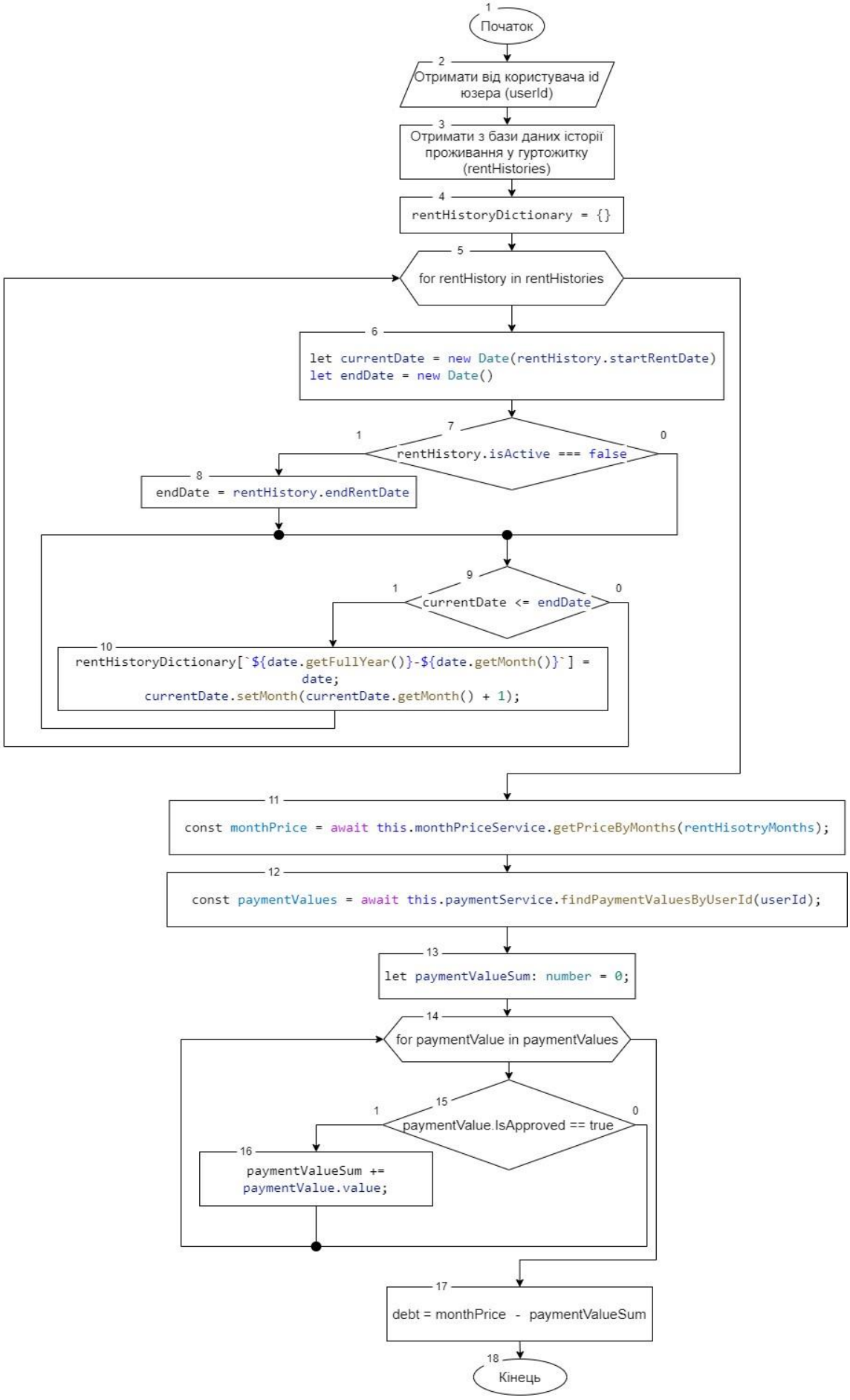


Рисунок 2.2 – Блок-схема алгоритму визначення боргу студента

Крок 9. Здійснити перевірку, чи змінна currentDate менше рівна endDate. Якщо так, то здійснюється перехід до кроку 10, якщо ні – до кроку 5.

Крок 10. Додати до rentHistoryDictionary дату (місяць проживання студента у гуртожитку). Також додати 1 місяць до дати currentDate.

Крок 11. Записати у змінну monthPrice ціну місяців, що прожили студенти.

Крок 12. Записати у змінну paymentValues проплати студента.

Крок 13. Записати у змінну paymentValueSum число 0.

Крок 14. Запустити цикл, який буде перебирати елементи списку проплат студента (paymentValues). Якщо список закінчився – перейти до кроку 17, якщо ні – до кроку 15.

Крок 15. Здійснити перевірку, чи змінна paymentValue.IsApproved дорівнює true. Якщо так, то здійснюється перехід до кроку 16, якщо ні – до кроку 14.

Крок 16. Додати до paymentValueSum значення змінної paymentValue.value.

Крок 17. Визначати суму боргу для студента, а саме записати у змінну debt різницю ціни за прожиті місяці (monthPrice) та суми проплат студента (paymentValueSum).

Крок 18. Кінець.

Таким чином відбувається розрахунок боргу для студента.

2.3 Розробка методу та алгоритму генерації документів з поточною інформацією

Генерація документів для управління гуртожитку та для взаємодії з гуртожитком, що мають у собі поточну інформацію є надзвичайно важливим функціоналом, оскільки дають можливість автоматизувати процес створення документів. Це надзвичайно сильно покращить ефективність управління гуртожитком та покращить якість взаємодії студента з гуртожитком, оскільки більше не буде потрібно йти у кабінет до коменданта за підписом, згенерувати документ можна буде дистанційно та за декілька секунд.

На даний момент методи аналоги генерації документів з поточною інформацією дозволяють створювати документи на основі інформації, що відправляються за допомогою форм. Тобто для кожної генерації документа потрібно заповнювати нову форму. Також для генерації документу потрібно розробити шаблон, що буде містити ключові слова. Ці ключові слова можна буде замінити на необхідну інформацію та на необхідні картинки. Найкращим механізмом отримання поточної та коректної інформації буде отримання інформації з профілів користувачів. Користувачу потрібно буде лише один раз заповнити всю необхідно інформацію для створення документів у профілі. Таким чином розроблюваний метод є кращим за аналоги, оскільки повністю автоматизує процес генерації документу. Розроблюваний метод буде використовувати інформацію з профілів користувачів та замінювати ключові слова у шаблоні на потрібну інформацію або картинки. Блок-схема алгоритму генерації документів з поточною інформацією наведена на рисунку 2.3.

Крок 1. Початок.

Крок 2. Отримати від користувача Id юзера, що хоче згенерувати шаблон та записати Id юзера у змінну userId.

Крок 3. Отримати від користувача назву стратегії документа та записати значення у змінну documentStrategyName.

Крок 4. Записати у змінну selectedDocumentStrategy значення null.

Крок 5. Запустити цикл, який буде перебирати елементи списку стратегій генерації документу (ducomentStrategies). Якщо список закінчився – перейти до кроку 8, якщо ні – до кроку 6.

Крок 6. Здійснити перевірку, чи змінна documentStrategyName дорівнює полю Name об'єкта documentStrategy. Якщо так, то здійснюється перехід до кроку 7, якщо ні – до кроку 5.

Крок 7. Присвоїти змінній selectedDocumentStrategy об’єкт у змінній documentStrategy.

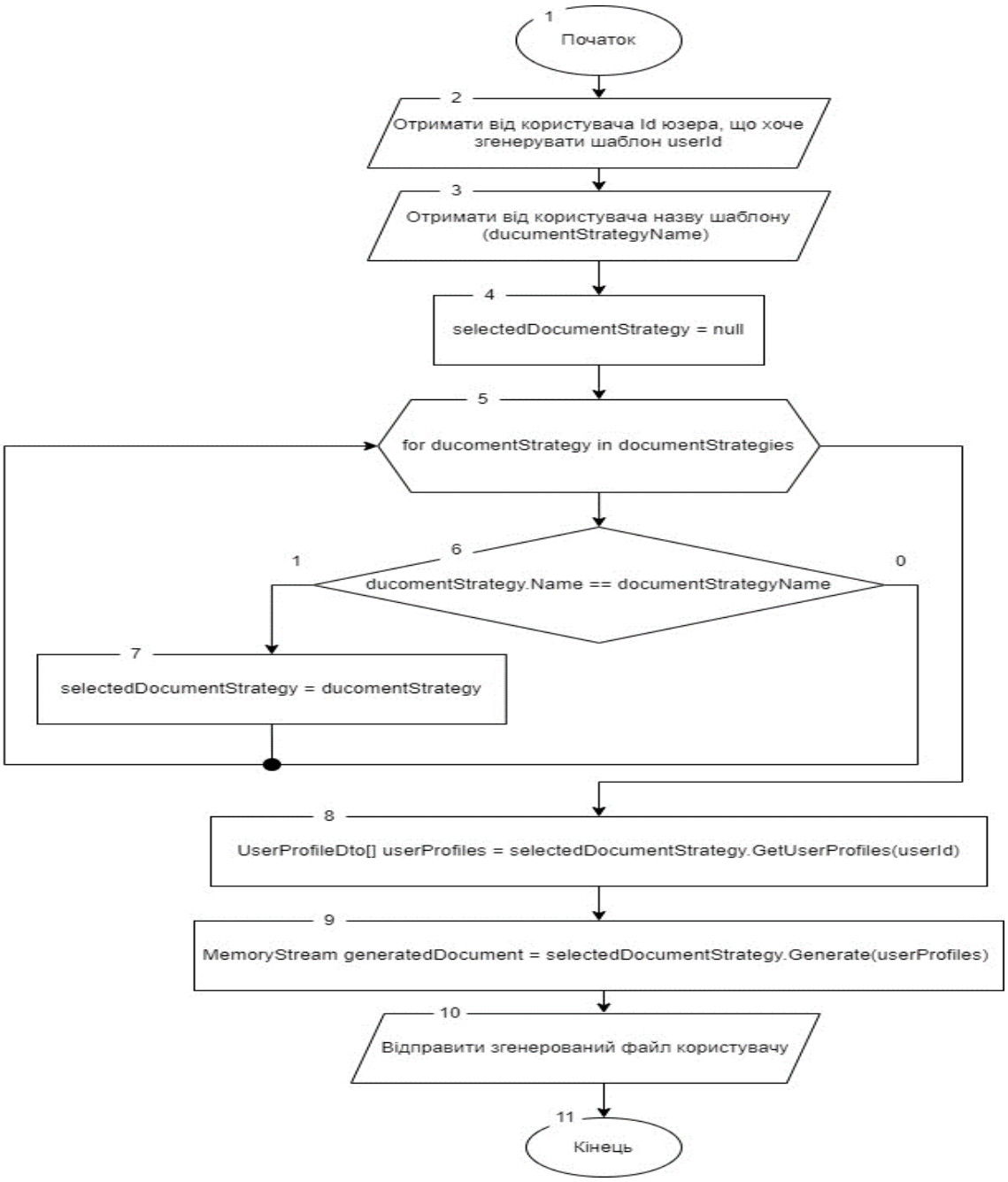


Рисунок 2.3 – Блок-схема алгоритму генерації документа з поточною інформацією

Крок 8. Присвоїти змінній userProfiles профілі користувачів, що потрібні для генерації документа.

Крок 9. Присвоїти змінній generatedDocument згенерований документ.

Крок 10. Відправити згенерований документ користувачу.

Крок 11. Кінець.

Таким чином відбувається генерація документів з поточною інформацією.

2.4 Розробка структури графічного інтерфейсу веб-застосунку для підвищення ефективності процесу управління гуртожитком університету

Графічний інтерфейс було реалізовано як веб-інтерфейс. Цей вид інтерфейсу повинен мати кнопки, випадаюче меню та інші елементи, що допомагають користувачу взаємодіяти з системою. Користувач може переходити по різним сторінкам, додавати, видаляти та змінювати інформацію або елементи на сторінках. Інтерфейс повинен не перенавантажувати користувача інформацією. Кожна веб-сторінка застосунку повинна бути зручною та зрозумілою. Також графічний інтерфейс не повинен втомлювати користувача, оскільки керуючий персонал буде проводити багато часу користуючись застосунком у робочі дні, доцільно використовувати кольори, що є нейтральними та не втомлюють користувача. Буде розроблений стандартний веб-інтерфейс, що буде включати в себе такі елементи:

1. Шапка (Header) – це верхній розділ сторінки або веб-сайту, який зазвичай містить важливу інформацію та функціональність для користувача. Вона розташована зверху сторінки і може містити такі елементи як логотип або назву бренду, меню навігації, пошукову панель, посилання на обліковий запис користувача;

2. Меню навігації (Navigation Menu) – це набір посилань або кнопок, які дозволяють користувачам легко переходити між різними розділами або сторінками веб-сайту чи веб-застосунку. Воно розташоване зазвичай у верхній частині сторінки веб-застосунку, нижче від шапки, і може бути відображене горизонтально або вертикально;

3. Основний вміст (Main Content) – це частина сторінки веб-сайту, де розміщена основна інформація, контент або функціональність, яка призначена для користувачів. Ця область містить важливий вміст, який зазвичай відображається на головній частині сторінки і який користувачі переглядають або з ним взаємодіють;

4. Підвал (Footer) – це нижня частина сторінки веб-застосунку або веб-сайту, яка зазвичай містить інформацію та посилання, що не є основною

частиною контенту, але є важливою для користувачів. Підвал розташовується внизу сторінки.

Головне вікно застосунку для студента та керуючого персоналу різне, оскільки студент та керуючий персонал будуть мати різний функціонал. На рисунку 2.4 зображена структурна схема інтерфейсу головного вікна застосунку для студента.

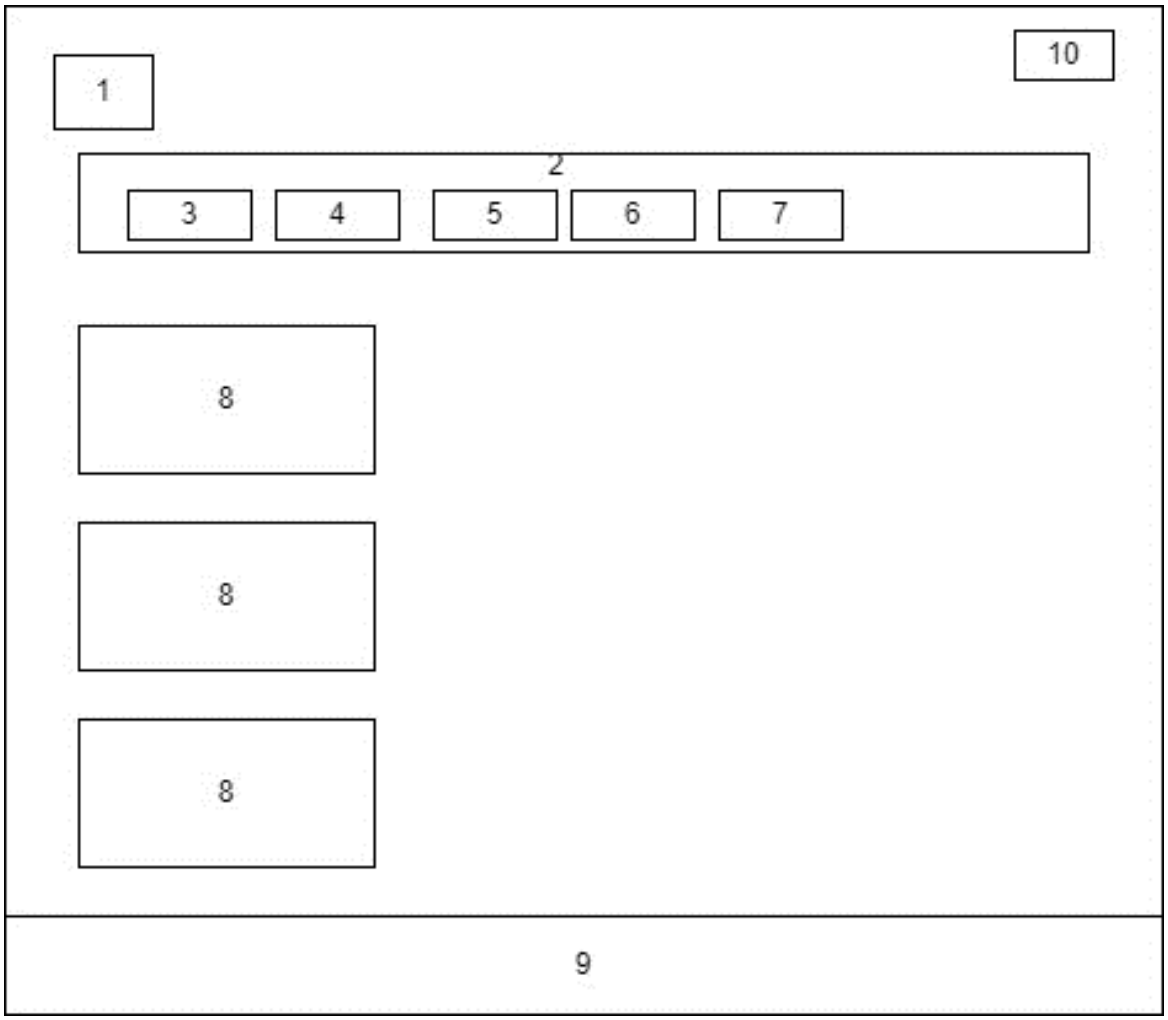


Рисунок 2.4 – Структурна схема інтерфейсу головного вікна застосунку для студента

Основні елементи головного вікна застосунку для студента:

1. Логотип.
2. Меню навігації.
3. Кнопка переходу на головну сторінку.

4. Кнопка переходу на сторінку проплати.
5. Кнопка переходу на сторінку коменданта.
6. Кнопка переходу на сторінку профілю.
7. Кнопка переходу на сторінку «Заява на поселення».
8. Новина університету.
9. Підвал.
10. Кнопка виходу.

На рисунку 2.5 зображена структурна схема інтерфейсу головного вікна застосунку для керуючого персоналу.

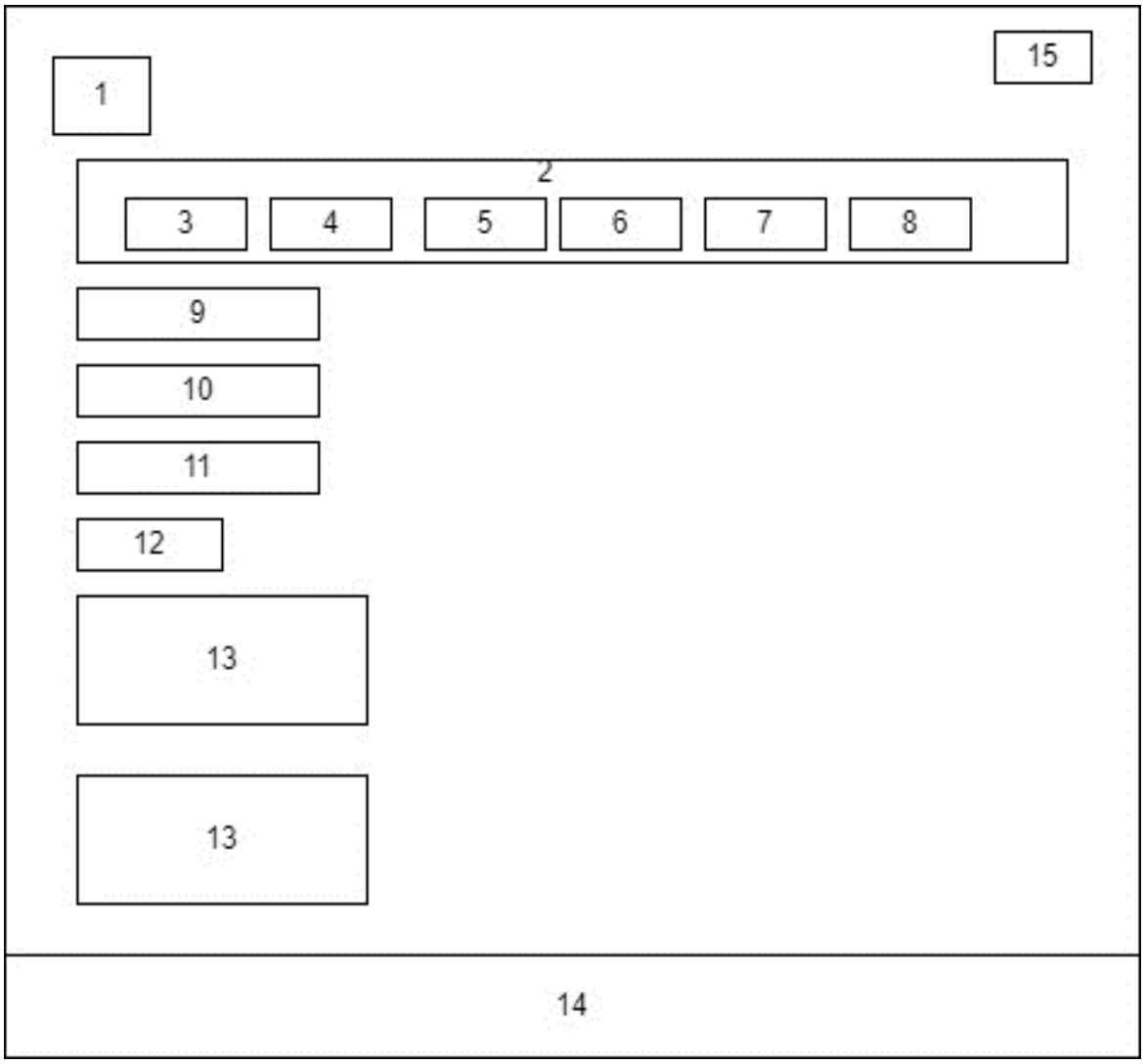


Рисунок 2.5 – Структурна схема інтерфейсу головного вікна застосунку для керуючого персоналу

Основні елементи графічного інтерфейсу головного вікна застосунку для керуючого персоналу:

1. Логотип.
2. Меню навігації.
3. Кнопка переходу на головну сторінку.
4. Кнопка переходу на сторінку списку студентів.
5. Кнопка переходу на сторінку реєстрації користувачів.
6. Кнопка переходу на сторінку кімнати.
7. Кнопка переходу на сторінку профілю.
8. Кнопка переходу на сторінку «Ціна проживання за місяць».
9. Поле для вводу заголовку новини.
10. Поле для вводу тіла новини.
11. Поле для вводу автора новини.
12. Кнопка додати новину.
13. Новина.
14. Підвал.
15. Кнопка виходу.

Головною перевагою веб-інтерфейсу є те, що користувач в більшості випадків знайомий з таким видом інтерфейсу і користувачу не потрібно буде витрачати час на додаткове ознайомлення з інтерфейсом застосунку. Також для веб-інтерфейсу існує велика кількість різного інструментарію, що дозволяє розробити інтерфейс більш зручним та привабливим. Веб-інтерфейс буде доступним через веб-браузер, що дає можливість користуватися застосунком для підвищення ефективності процесу управління гуртожитку університету з різних пристроїв. Не потрібно буде витрачати додаткові ресурси на додавання можливості користуватися застосунком з різних пристроїв, що зробить розробку застосунку дешевшим. Також доцільно не використовувати анімації у веб-інтерфейсі застосунку, оскільки вони будуть відволікати користувача і розробка анімацій вимагає додаткових та значних ресурсів. Доцільно використовувати максимально простий дизайн для не перевантаження користувача.

2.5 Висновки

У першому підрозділі було розроблено метод та алгоритм розпізнання суми проплати студента. Було визначено, що доцільно розпізнавати суму проплати лише у певній ділянці картинки, та отримувати потрібну інформацію з розпізнаного тексту шляхом пошуку унікального паттерну потрібної інформації за допомогою Regex. Також було визначено, що доцільно використовувати паттерн стратегії, оскільки це дає можливість розпізнавати різні проплати з різних банків. Також було розроблено блок-схему алгоритму розпізнання суми проплати.

У другому підрозділі було розроблено метод та алгоритм розрахунку боргу студента. Було визначено, що борг студента визначається різницею всіх цін прожитих місяців студента у гуртожитку та проплат студента, що є підтвердженими, оскільки розпізнання тексту не є стовідсотково надійним. Було розроблено блок-схему алгоритму розрахунку боргу студента.

Також було розроблено метод генерації документів з поточною інформацією. Було вирішено, що поточну інформацію найдоцільніше брати з профілів користувачів. Для генерації документу користувачу буде потрібно заповнити свій профіль та завантажити підписи лише один раз, що робить генерацію документів автоматизованим процесом. Також було розроблено блок-схему алгоритму генерації документів з поточною інформацією.

Також було розроблено структурну схему інтерфейсу головного вікна застосунку для студента та для керуючого персоналу, оскільки керуючий персонал та студент будуть мати різний функціонал при взаємодії із застосунком для підвищення ефективності процесу управління гуртожитку. Також було вирішено, що доцільно не використовувати анімації при взаємодії користувача з інтерфейсом. Також було вирішено, що потрібно використовувати максимально прості елементи та простий дизайн, щоб не перевантажувати користувача непотрібною візуальною інформацією.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ ГУРТОЖИТКОМ УНІВЕРСИТЕТУ

3.1 Варіантний аналіз і обґрунтування вибору мови програмування

Вибір мови програмування є надзвичайно важливим при розробці програмного забезпечення, оскільки це надзвичайно сильно впливає на швидкість розробки та на простоту супроводження, а отже і на ціну проекту. Мова програмування повинна бути сучасною та популярною, щоб інші розробники могли без проблем почати супруводжувати проект. Мова програмування повинна відповідати вимогам для розробки застосунку для підвищення ефективності процесу управління гуртожитком. Застосунок повинен бути легким до внесення нового функціоналу, легко масштабованим, та повинен легко розгортатися. Для виконання цього мова програмування повинна бути об’єктно-орієнтованою та мати потужні фреймворки та інструментарій для розробки.

Розглянемо одні з найпопулярніших мов програмування:

1. TypeScript – це мова програмування, яка використовується для розробки веб-застосунків та програмного забезпечення. Вона є розширенням мови JavaScript і надає можливість розробникам використовувати статичну типізацію для покращення надійності та продуктивності коду. Однією з основних переваг TypeScript є можливість визначати типи даних для змінних, функцій та об'єктів, що допомагає виявляти помилки на етапі розробки та полегшує рефакторинг коду [11];

2. JavaScript – це високорівнева, інтерпретована мова програмування, яка широко використовується для розробки веб-застосунків та взаємодії зі стороною клієнта на веб-сторінках. Вона надає можливість динамічно створювати і змінювати вміст веб-сторінок, реагувати на події користувачів та спілкуватися з веб-серверами. JavaScript є невід'ємною частиною сучасної розробки веб-застосунків і використовується в поєднанні з HTML і CSS для створення інтерактивних і динамічних веб-інтерфейсів [12];

3. Java – це потужна і універсальна об'єктно-орієнтована мова програмування, яка використовується в різних галузях розробки. Вона відзначається високою надійністю, переносимістю та широкою підтримкою спільноти розробників. Java використовується для створення різноманітних застосунків, включаючи мобільні застосунки, веб-застосунки, великі корпоративні системи та вбудовані програми. Вона підтримує багатофункціональність та має велику кількість бібліотек і фреймворків [13];

4. C++ – це потужна і многозадачна мова програмування, що володіє широким спектром застосувань. Вона поєднує в собі можливості низькорівневої мови програмування, як C, з об'єктно-орієнтованим програмуванням, що дозволяє створювати як швидкі і ефективні системи, так і складні застосунки з використанням об'єктно-орієнтованого підходу [14].

Результати порівняння даних мов програмування зведено до таблиці 3.1.

Таблиця 3.1 – Порівняння мов програмування

Критерій	TypeScipt	JavaScript	Java	C++
Об'єктно-орієнтована	1	1	1	1
Популярність	1	1	1	1
Безпека	1	0	1	0
Присутність фреймворків	1	1	1	0
Простота мови	1	1	0	0
Швидкість	1	1	1	1
Всього	6	5	5	3

За результатами порівняння, що представлені на таблиці 3.1 можна зробити висновок, що використовувати мову програмування TypeScipt найбільш доцільно, оскільки ця мова програмування відповідає всім вимогам, що потрібні для розробки застосунку для підвищення ефективності процесу управління гуртожитку для університету. Також для TypeScript присутній потужний фреймворк NestJS. За допомогою фреймфорку NestJS буде розроблятися основний функціонал застосунку.

3.2 Вибір середовища розробки та СУБД

Середовище розробки є надзвичайно важливим інструментом при розробці великих програмних продуктів. Оскільки середовище розробки надає інструментарій для відлагодження застосунку, що надзвичайно сильно пришвидшує розробку. Використання середовища розробки є доцільним при розробці власного застосунку для підвищення ефективності управління гуртожитку для університету, бо розроблюваний застосунок є великим і розробка без середовища розробки буде повільною та складною. Середовище розробки повинно бути популярним, забезпечувати необхідний функціонал для розробки застосунку. Також середовище розробки повинно підтримувати мову програмування TypeScript, бо це основна мова програмування, що буде використовуватись при розробці. Найбільш популярними серед середовищ розробки на мові програмування TypeScript є Visual Studio, WebShtorm та Visual Studio Code.

Visual Studio – це потужне інтегроване середовище розробки (IDE), розроблене компанією Microsoft, яке надає розробникам зручність та ефективність під час створення програмного забезпечення для різних платформ і мов програмування [15]. Це середовище розробки ідеально підходить для створення різних видів застосунків, від десктопних програм до веб-сайтів та мобільних застосунків. Visual Studio також надає інструменти для тестування та спільної роботи в команді. Незалежно від рівня навичок розробника, Visual Studio створює зручні умови для продуктивної розробки програмного забезпечення на будь-якому етапі проекту. Visual Studio має багато розширень, що дає можливість підтримувати багато різних мов програмування. Також Visual Studio має потужні розширення, що допомагають писати clean code. Visual Studio надає потужний функціонал відлагоджування, що надзвичайно сильно пришвидшує розробку застосунку та допомагає визначити помилки. Однією з найбільших проблем Visual Studio є швидкість роботи, оскільки це середовище розробки надзвичайно потужне та має великий функціонал, то робота з цим середовищем вимагає багато ресурсів комп’ютера. Інтерфейс середовища розробки Visual Studio наведено на рисунку 3.1.

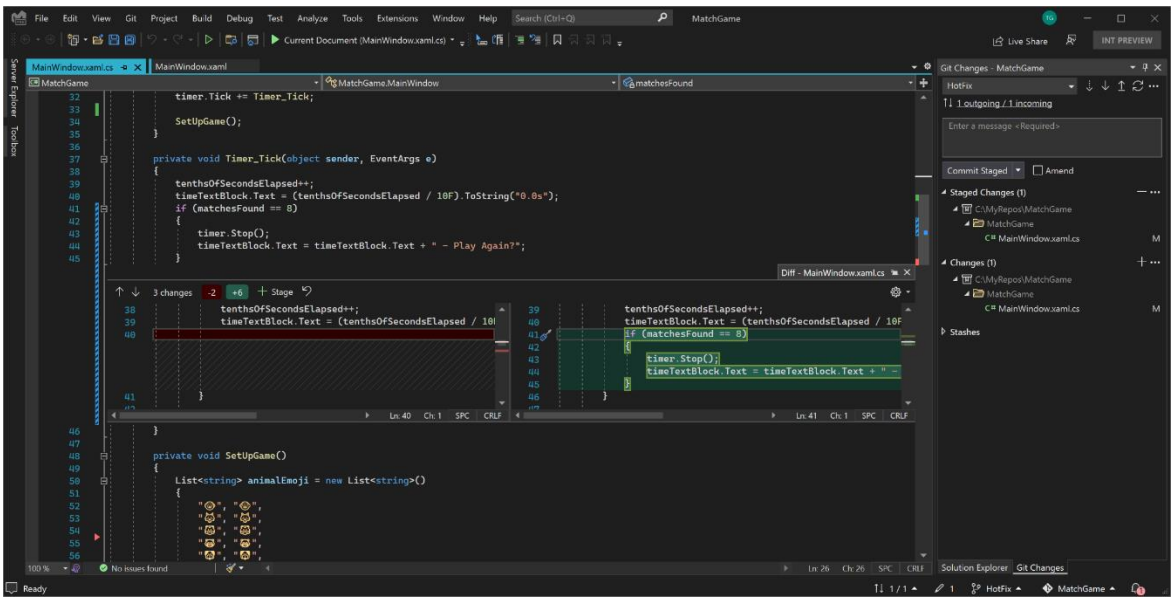


Рисунок 3.1 – Інтерфейс середовища розробки Visual Studio

WebShtorm – це інтегроване середовище розробки (IDE), розроблене компанією JetBrains спеціально для веб-розробки [16]. Ця платформа надає розробникам зручний та потужний інструментарій для створення веб-застосунків, включаючи веб-сайти, мобільні застосунки з використанням HTML, CSS та JavaScript. Основні особливості WebStorm включають розширений редактор коду, автоматичну перевірку синтаксису та підказки, що допомагають уникати помилок. Також надає інтеграцію з системами контролю версій, такими як Git, що сприяє ефективній роботі в команді. Інтерфейс середовища розробки WebShtorm наведено на рисунку 3.2.

Рисунок 3.2 – Інтерфейс середовища розробки WebShtorm

Visual Studio Code – це безкоштовне та надзвичайно популярне інтегроване середовище розробки (IDE), розроблене компанією Microsoft. Відмінною рисою VS Code є його універсальність і відкритість для різних типів розробок, незалежно від мови програмування чи платформи [17]. Завдяки великій кількості розширень, VS Code підтримує практично будь-яку мову програмування та фреймворк. Воно має потужний редактор коду з підсвічуванням синтаксису, автодоповненням, інтегрованими інструментами для роботи з Git і підтримкою віртуальних середовищ. VS Code також надає можливість налаштувати робоче середовище для вас, встановивши лише необхідні розширення та інструменти. Також середовище розробки Visual Studio Code добре оптимізовано, що дозволяє навіть слабким комп'ютерам працювати з ним. Інтерфейс середовища розробки Visual Studio Code наведено на рисунку 3.3.

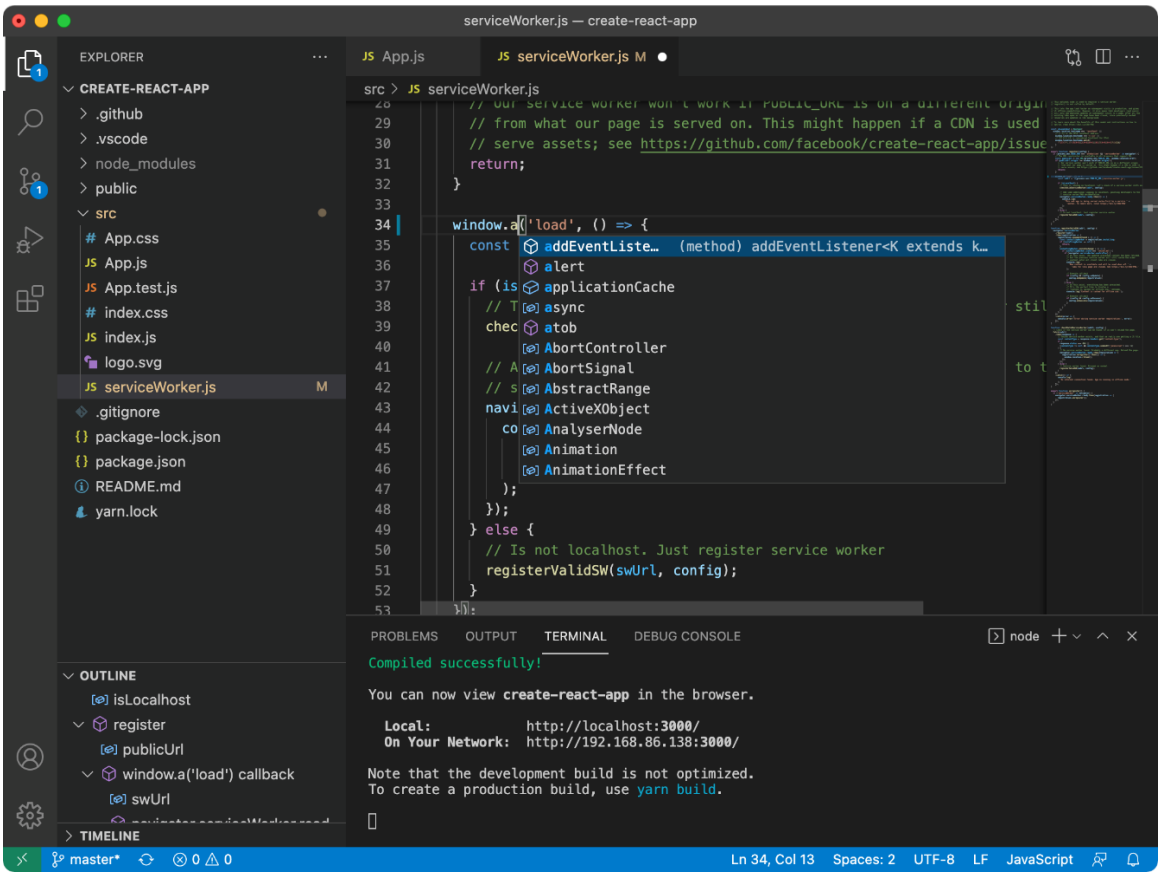


Рисунок 3.3 – Інтерфейс середовища розробки Visual Studio Code

Результати порівняння середовищ розробки зведено до таблиці 3.2.

Таблиця 3.2 – Порівняння інтегрованих середовищ розробки

Функція/Характеристика	Visual Studio	WebStorm	Visual Studio Code
Вартість	0	0	1
Підтримка TypeScript	1	1	1
Інтеграція з системами контролю версій	1	1	1
Розширюваність через плагіни	1	0	1
Підтримка веб-розробки	1	1	1
Підтримка хмарових рішень	1	0	1
Всього	5	3	6

За результатами порівняння доцільно використовувати Visual Studio Code як середовище розробки, оскільки це середовище забезпечує весь потрібний функціонал для розробки застосунку для підвищення ефективності управління гуртожитку університету та також є кращим за аналоги.

Застосунок повинен бути доступним через інтернет та повинен обробляти запити користувачів. Обробка запитів має у собі функцію збереження, редагування, отримання різних даних. Для вирішення цієї проблеми потрібно використовувати базу даних. База даних є одним з найважливіших компонентів застосунку, оскільки без цього не можлива робота застосунку. База даних повинна бути швидкою та бути толерантною до змін, оскільки у процесі розробки, об’єкти, що зберігаються можуть мінятись. Доцільно розглядати бази даних, що є нереляційними, оскільки вони забезпечують збереження інформації в об’єкти, що дозволяє сильно підвищити швидкість розробки та полегшити супровід проекту. Найпопулярнішими нереляційними базами даних є: MongoDB, CouchDB, RethinkDB.

MongoDB – це нереляційна база даних, яка використовує документоорієнтовану модель даних для зберігання і обробки інформації [18]. У MongoDB дані зберігаються у вигляді JSON-подібних документів, які можуть бути вкладені один в одного. Це робить її дуже гнучкою та підходить для

сховища різноманітних даних. MongoDB відома своєю здатністю масштабуватися горизонтально, що дозволяє обробляти великі обсяги даних і високі навантаження. Вона також підтримує автоматичну реплікацію для забезпечення високої доступності і може використовувати шари для розділення даних за логічними категоріями.

CouchDB – це нереляційна база даних, яка відзначається своєю документоорієнтованою моделлю зберігання даних і відкритим джерелом [19]. Її основна концепція – це зберігання даних у JSON-подібних документах, які можуть бути вкладені один в одного. Це робить її ідеальним вибором для проектів, де потрібно зберігати семі-структуровані або незалежні дані.

RethinkDB – це інноваційна нереляційна база даних, спроектована для роботи у реальному часі та для гнучкої роботи з даними [20]. Її основна відмінність – це здатність автоматично оновлювати результати запитів при зміні даних, надаючи можливість роботи з "живими" даними.

Результати порівняння даних систем управління базами даних зведено до таблиці 3.3.

Таблиця 3.3 – Порівняння систем управління базами даних

Критерій	MongoDB	CouchDB	RethinkDB
Модель даних	1	1	1
Можливість відслідковування змін	1	0	1
Доступність шару сховища	1	1	0
Масштабованість	1	1	1
Гнучкість схеми	1	1	1
Доступність реплікації	1	1	1
Швидкість	1	0	1
Популярність	1	0	0
Всього	8	5	6

За результатами порівняння, наведених у таблиці 3.3, було визначено, що доцільно використовувати нереляційну базу даних MongoDB, оскільки вона забезпечує всі потреби застосунку для підвищення процесу управління гуртожитком університету. Також MongoDB є популярнішою базою даних ніж інші аналоги, що означає, що є більше навчальних матеріалів для MongoDB.

Отже, за результатами порівняння найпопулярніших середовищ розробки, що використовують мову TypeScript. Доцільно використовувати Visual Studio Code, оскільки це середовище розробки повністю забезпечує всім необхідним для розробки застосунку для управління гуртожитком. Також найкращим вибором нереляційної бази даних є MongoDB, оскільки база даних забезпечує гнучкість при розробці застосунку. Також база даних MongoDB краща за існуючі аналоги.

3.3 Розробка інтерфейсу програмного застосунку за допомогою React

Застосунок для збільшення ефективності процесу управління гуртожитку буде розроблятися з використанням веб-інтерфейсу, оскільки це найбільш поширений вид інтерфейсу і користувачу не потрібно багато практики або знань щоб користуватися даним видом інтерфейсу, бо цей вид інтерфейсу знайомий користувачу.

Веб-інтерфейс – це спосіб взаємодії користувача з програмними застосунками, сервісами або системами через веб-браузер. Веб-інтерфейс зазвичай включає в себе веб-сторінки, на яких розміщена інформація та елементи управління, такі як кнопки, форми, гіперпосилання та інші елементи, які користувач може використовувати для взаємодії з програмою або сервісом [21].

Стандартними елементами веб-інтерфейсу є кнопки, поля для вводу, випадні меню, текстові поля та картинки. Інтерфейс повинен бути зрозумілим і не перевантажувати користувача інформацією. Тому потрібно створювати декілька веб сторінок які будуть у собі мати лише ту інформацію, яка потрібна користувачу. Для цього потрібно зробити систему навігації по сторінкам. Також повинна бути можливість динамічної зміни інформації на екрані при взаємодії з

інтерфейсом (видалення або додавання об’єкту). Інтерфейс також повинен динамічно отримувати дані з серверу та їх відображати. Інтерфейс повинен бути зрозумілим та естетично приємним.

Розробка веб-інтерфейсу є надзвичайно складним процесом. Для вирішення вище згаданих проблем буде проблематично використовувати стандартний інструментарій (html, css, js). Застосунок є надзвичайно комплексним. Тому доцільно використовувати бібліотеку для розробки інтерфейсу. Існують різні бібліотеки, які використовуються для створення інтерфейсу користувача. Вони призначені для пришвидшення розробки і мають у собі вже існуючі рішення певних задач. Найвідоміші бібліотеки:

1. React – це JavaScript бібліотека для розробки інтерактивних користувацьких інтерфейсів (UI) на веб-сторінках. Вона була розроблена компанією Facebook і є однією з найпопулярніших бібліотек для створення веб-застосунків та веб-сайтів. React дозволяє розробникам створювати інтерфейси, які ефективно реагують на зміни стану застосунку та відображають їх на сторінці без перезавантаження всієї сторінки [22];

2. Angular – це фреймворк для розробки веб-застосунків. Він був розроблений і підтримується компанією Google і призначений для створення високоякісних та розширюваних веб-застосунків та веб-сайтів. Основними характеристиками якого є: компонентна архітектура, керування станом, розширена маршрутизація [23];

3. Vue – це прогресивний JavaScript фреймворк для розробки користувацьких інтерфейсів. Він створений для побудови веб-застосунків та односторінкових застосунків (SPA). Основними характеристиками є: легкість та компактність, компонентна архітектура, двостороннє зв'язування даних [24]. Також Vue є популярним фреймворком.

Створимо таблицю та порівняємо бібліотеки для розробки інтерфейсу користувача для веб-застосунків. Бібліотека для розробки інтерфейсу повинна бути швидкою, підтримувати мову TypeScript. Результат порівняння наведено у таблиці 3.4.

Таблиця 3.4 – Порівняльні характеристики бібліотек інтерфейсу

Критерій	React	Angular	Vue
Компонентна архітектура	1	1	1
Комплектація	1	0	1
Швидкість	1	0	1
Двостороннє зв'язування даних	1	1	1
Маршрутизація	1	1	0
Всього	5	3	4

На основі результатів таблиці можна зробити висновок, що React є найкращим рішенням для використання його як бібліотеки розробки графічного інтерфейсу користувача.

React дозволяє розробляти інтерфейс використовуючи компоненти, що дуже зручно, оскільки це дозволяє підвищити ефективність розробки та зменшити кількість написаного коду, бо код, що був написаний у компоненті можна використовувати у іншій компоненті. Також React дозволяє використовувати навігацію та переходити на інші сторінки застосунку при натисненні кнопки керування. У застосунку існує велика кількість компонент таких як: компонента боргу, компонента проплати, компонента профілю користувача, компонента заяви на проживання, компонента менеджменту студентів та інші компоненти. Розглянемо компоненту застосунку для менеджменту студентів. На рисунку 3.4 зображено код дочірньої компоненти менеджменту студента.

```
export default function UserProfileListCard(props: any) {
    const navigate = useNavigate();
    return (
        <div className='studentCardBody'>
            <p className='userName'>{props.userName}</p>
            <button className='controll-button'
                onClick={() => navigate(`${NavigationUrls.USER_PROFILE}/${props.userId}`)}>Профіль</button>
            <button className='controll-button'
                onClick={() => navigate(`${NavigationUrls.ADMIN_PAYMENT}/${props.userId}`)}>Проплата</button>
        </div>
    )
}
```

Рисунок 3.4 – Лістинг коду дочірньої компоненти менеджменту студента

На рисунку 3.4 зображено кнопоки керування для менеджменту студента, при натисненні на які, користувач буде перекинутий на іншу сторінку. Також посилання на яке переходить користувач містить у собі контекст, userId студента, що дозволяє відкривати сторінку користувачу з потрібною інформацією. Також навігаційні посилання є динамічними і вони отримуються з сервісу NavigationUrls, що дозволяє при зміні посилання змінити це посилання у одному місці застосунку, а не у декількох місцях. Оскільки студентів може бути багато, то потрібно використовувати декілька разів компоненту кнопок керування для менеджменту студента, тобто потрібно розробити керуючу компоненту. На рисунку 3.5 Зображено лістинг коду керуючої компоненти для менеджменту студентів.

```
export default function UserProfileList(props: any) {

    const [users, setUsers] = useState([]);
    const initializeData = async () => {
        usersService.getUsersList().then((result) => {
            console.log(result.data);
            setUsers(result.data);
        });
    };
    useEffect(() => {
        initializeData();
    }, []);
    return (
        <div>
            {users.map((user: any, i) => {
                // Return the element. Also pass key
                return (<UserProfileListCard key={i} userId={user.id} userName={user.userName} />)
            })}
        </div>
    )

}
```

Рисунок 3.5 – Лістинг коду компоненти для менеджменту студентів

На рисунку 3.5 зображена керуюча компонента для менеджменту студентів, що використовує компоненту кнопок керування для менеджменту студента. Це дозволяє динамічно створювати записи на інтерфейсі для кожного користувача. Користувачі, для відображення на інтерфейсі, отримуються за допомогою запитів на backend сервіс. З керуючої компоненти передається потрібна інформація для дочірніх компонент, що дозволяє відображати дочірні

компоненти з потрібно інформацією. Функція useEffect дозволяє виконувати код при першому заходу на сторінку або при перезавантаженні сторінки, що дозволяє ініціалізувати компоненту потрібною інформацією з backend сервісу. Взаємодія з backend сервісом відбувається за допомогою сервісів взаємодії, такі сервіси присутні для кожної керуючої компоненти, що потрібує інформацію з backend сервісу. На рисунку 3.6 зображено лістинг коду сервісу взаємодії з backend сервісом для керуючої компоненти менеджменту студента.

```
class UsersService {
    public async getUsersList() {
        const usersUrl: any = process.env.REACT_APP_USERS_URL;
        const result = await apiRequestService.makeRequest(AxiousRequestMethod.get, usersUrl);
        return result;
    }
}
const usersService = new UsersService();
export default usersService;
```

Рисунок 3.6 – Лістинг коду сервісу взаємодії з backend сервісом для компоненти менеджменту студента

Взаємодія з backend сервісом відбувається за допомогою Http запитів. Сервіси взаємодії можуть відправляти різні запити такі як: GET, POST, DELETE, PUT, що дозволяє отримувати, додавати, змінювати, видаляти потрібну інформацію з backend сервісу. При помилці відправки запиту на backend сервіс, застосунок буде коректно обробляти помилки та логувати помилку у консоль розробника.

Отже у підрозділі було визначено, що доцільно використовувати React як бібліотеку для створення інтерфейсу застосунку, оскільки бібліотека React повністю забезпечує всім необхідним для розробки інтерфейсу. Також було визначено основну структуру розробки компонент інтерфейсу застосунку, що складаються з керуючого компонента, дочірніх компонент, що використовує керуючий компонент та з сервісу взаємодії з backend сервісом. Приклад було наведено за допомогою компоненти менеджменту студента.

3.4 Програмна реалізація застосунку для підвищення ефективності процесу управління гуртожитку університету

Реалізація програмного функціоналу застосунку для підвищення ефективності процесу управління гуртожитку для університету полягає у створенні наступного функціоналу:

- функціонал розпізнання суми проплати з чеків, що завантажуються студентом;
- функціонал розрахування боргу студента;
- функціонал генерації документів з використанням поточної інформації;
- функціонал створення новин університету;
- функціонал менеджменту інформації про студента;
- функціонал встановлення ціни проживання за місяць.

Застосунок було реалізовано з використанням мікросервісної архітектури, тобто було реалізовано декілька сервісів, що відповідають за свій функціонал, а саме:

IdentityService – сервіс авторизації та аутентифікації користувачів, надає доступ або забороняє доступ до даних на інших сервісах, також надає функціонал для розпізнання суми проплати, що завантажується студентом, написаний на мові C#;

WordGeneratorService – сервіс, що надає можливість генерувати документи по шаблону з використанням поточної інформації, написаний на мові C#;

ApiService – серверна частина застосунку, написаний на мові TypeScript;

FrontedService – сервіс графічного інтерфейсу користувача, що написаний з використанням React, написаний на мові TypeScript.

При ініціалізації програмних продуктів, першим чином ініціалізуються сервіси завдяки dependency injection, здійснюється перевірка підключенння до бази даних. Після запуску фреймворку сервіси готові приймати запити від користувачів. Лістинг коду ініціалізації для сервісу ApiService наведено на рисунку 3.7.

```
import { ClassSerializerInterceptor, ValidationPipe } from '@nestjs/common';
import { NestFactory, Reflector } from '@nestjs/core';
import { AppModule } from './app/app.module';
// import { runInCluster } from '@common/utils/run-in-cluster';

async function bootstrap() {
  process.env.TZ = 'Etc/Universal';
  const app = await NestFactory.create(AppModule);
  app.enableCors();
  app.useGlobalPipes(new ValidationPipe({ skipMissingProperties: true }));
  app.useGlobalInterceptors(new ClassSerializerInterceptor(app.get(Reflector)));
  const port = process.env.SERVER_PORT;
  await app.listen(port, () => {
    console.log(`Server is running at http://localhost:${port}/api/docs`);
  });
}
bootstrap();
```

Рисунок 3.7 – Лістинг коду ініціалізації сервісу ApiService

Користувач при взаємодії з застосунком для підвищення ефективності процесу управління гуртожитку взаємодіє з інтерфейсом, що був згенерований FrontedService. Для отримання або редагування інформації FrontedService взаємодіє з ApiService завдяки Http запитам. При кожній генерації сторінки або певної дії користувача FrontedService відправляє запит на ApiService, що містить у собі токен для авторизації та аутентифікації користувача. ApiService перевіряє цей токен та вирішує чи надавати доступ до ресурсу. Токен отримується користувачем при вході у застосунок, а саме введенням логіну та паролю у сервісі IdentityService. Взаємодія між різними сервісами відбувається з використанням протоколу OpenId. Тобто для взаємодії між різними сервісами потрібно ввести логін та пароль один раз і після отримання токену можна відправляти цей токен у різні сервіси. Використання проколу OpenId при розробці застосунку для підвищення ефективності процесу управління гуртожитку допомагає реалізувати авторизацію та аутентифікацію для мікросервісів застосунку, що надзвичайно сильно пришвидшує швидкість розробки застосунку та зменшує вартість розробки та підтримки застосунку. Валідація токену та отримання потрібної інформації з токену зображена на рисунку 3.8.

```
private validateIsHavePermissionAndGetUserAndRole(token: string, context: ExecutionContext) {
    const decodedToken = jwt.decode(token, { complete: true });
    const tokenRole = decodedToken['payload']['role'];
    const roles = this.reflector.get<string[]>('roles', context.getHandler());
    const scopes = this.reflector.get<string[]>('scopes', context.getHandler());
    if (this.authorizeToken(decodedToken, tokenRole, roles, scopes) == false) {
        console.log('forbidden')
        throw new ForbiddenException('Forbidden');
    }
    const tokenUserId = decodedToken['payload']['sub'];
    return {
        role: tokenRole,
        userId: tokenUserId
    }
}
```

Рисунок 3.8 – Лістинг валідації та отримання необхідної інформації з токену

Токен відправляється та отримується з хедеру Authorization. Для валідації токену спочатку токен декодується. З декодованого токену потім отримується роль. Потім перевіряється чи присутня роль, що відправив користувач у токені у дозволених ролях для контролеру. Також перевіряється чи присутній відісланий користувачем у токені scope у дозволених scopes контролеру. Якщо роль або scope присутні, то користувачу дається доступ до ресурсу, інакше відправляється помилка Forbidden. Потім повертається у реквест об’єкт, що має у собі роль користувача та userId користувача.

Користувач взаємодіє з застосунком через інтерфейс, що згенерований сервісом FrontedService з використанням бібліотеки React. Кожна сторінка у React складається з певних компонентів. Користувач може взаємодіяти з інтерфейсом через кнопки або поля вводу. Для реалізації інтерфейсу функціоналу для створення та показу новин для гуртожитку, було створено дві компоненти, а саме: компонента головного вікна новини та компонента новини. Компонента головного вікна новини може мати у собі декілька компонент новини. Також при відкритті сторінки або оновленні головного вікна застосунку завантажуються новини з backend сервісу. Лістинг розмітки компоненти

головного вікна застосунку, що містить новини зображено на рисунку 3.9.

```
const renderNotifications = () => {
    return (
        <div>
            {notifications.map((notification, i) => {
                return (<NotificationCardComponent key={i} props={notification} />)
            })}
        </div>
    );
}
const renderAddNotificationForComendant = () => {
    if (role === 'comendant') {
        return (<AddNotificationComponent setNotifications={setNotifications} />)
    }
    return (
        <div></div>
    )
}
return (
    <div>
        <div className='blackLine'></div>
        {renderAddNotificationForComendant()}
        {renderNotifications()}
    </div>
)
```

Рисунок 3.9 – Лістинг коду розмітки компоненти головного вікна застосунку, що містить новини

Лістинг коду розмітки компоненти головного вікна застосунку містить три функції що генерують html. Функція return (), це головна функція компоненти, що використовує інші функції генерації html. Функція renderNotifications займається генерацією html коду для створення новин і використовує компоненту NotificationCardComponent.

Функція renderAddNotificationForComendant виконує генерацію компоненти для додавання нової новини. Функціональність додавання нової новини буде доступна лише користувачу з роллю comendant, тобто для користувачів з іншими ролями компонента AddNotificationComponent генерувати html code та відображатися на інтерфейсі не буде. Також компоненті AddNotificationComponent передається функція setNotifications, за допмогою якої при додаванні нової новини, новини на головній сторінці будуть оновлюватися. Також при першому відкритті сторінки головного вікна застосунку або при перезавантаженні сторінки потрібно ініціалізувати дані та завантажити необхідну інформацію з backend сервісу. На рисунку 3.10 зображено лістинг коду ініціалізації інформації для головного вікна застосунку.

```
useEffect(() => {
    initializeData();
}, []);
const initializeUserRole = async () => {
    const initialLoginState = await authService.getUserRole();
    setRole(initialLoginState);
}
const updateNotifications = async () => {
    const result = await notificationService.getNotifications();
    setNotifications(result.data);
}
```

Рисунок 3.10 – Лістинг коду ініціалізації інформації головного вікна застосунку

Ініціалізація інформації головного вікна відбувається з функції useEffect, що відповідає за виконання коду при першому відкритті застосунку та при перезавантаженні сторінки. Функція useEffect викликає функцію initializeData, що викликає функцію initializeUserRole та функцію updateNotifications. Функція initializeUserRole ініціалізує роль користувача, що відкрив сторінку, шляхом зчитування ролі з токену. Також функція updateNotifications отримує з backed сурвісу всі новини про гуртожиток, ця функція виконується також при додаванні нової новини керуючим персоналом.

Компонента головного вікна використовує компоненту новини. Після додання нової новини сторінку потрібно обновити. Після оновлення сторінки з’явиться новина, що була додана керуючим персоналом. Також функціонал додавання новини доступний лише керуючому персоналу. У коді присутня перевірка чи користувач має роль коменданта, якщо так то фунціонал буде показано користувачу, якщо ні то це буде пустий блок. У новині вказується заголовок, тіло, дата та автор новини. Дата новини генерується при створенні новини і це повинна бути поточна дата. Також новина отримує всю необхідну інформацію з керуючого компоненту і вся інформація передається за допомогою змінної props. Також компонента новини має у собі розмітку html та css. За допомогою css новині надаються стилі відображення, що покращує естетичне сприйняття новини на інтерфейсі застосунку. На рисунку 3.11 зображено лістинг коду компоненти новини.

```
export default function NotificationCardComponent(props: any) {

    return (
        <div className='notificationBody'>
            <h1>{props.props.header}</h1>
            <p>{props.props.body}</p>
            <div className='notification-dateAndAuthor'>
                <p>{props.props.date}</p>
                <p>{props.props.author}</p>
            </div>
        </div>
    );
}
```

Рисунок 3.11 – Лістинг коду компоненти новини

Також компоненти для нотифікацій повинні взаємодіяти з серверною частиною. Вони відправляють запити на серверну частину для добавлення новини або отримання списку новин. Також запити йдуть на API service і тому змінна з середовища програми називається REACT_APP_DIPLOMNA_API_URL_BASE. Запит на добавлення нової новини використовує метод post, а запит на отримання новин використовує метод get. На рисунку 3.12 зображено лістинг коду взаємодії.

```
class NotificationService {
    public async addNotification(notification: any) {
        await apiRequestService.makeRequest(AxiousRequestMethod.post,
            `${process.env.REACT_APP_DIPLOMNA_API_URL_BASE}/notification`, notification);
    }
    public async getNotifications() {
        const result = await apiRequestService.makeRequestAnonim(AxiousRequestMethod.get,
            `${process.env.REACT_APP_DIPLOMNA_API_URL_BASE}/notification`);
        return result;
    }
}
const notificationService = new NotificationService();
export default notificationService;
```

Рисунок 3.12 – Лістинг коду сервісу новин взаємодії між фронтендом та бекендом

Функціональність новин для університету повинна включати можливість отримання всіх новин для буть-якого користувача та можливість додати нову новину лише для керуючого персоналу. Тому для контролеру для додавання нової новини потрібно добавити guard щоб тільки користувачі з роллю коменданта могли додавати нову новину, а для отримання всіх новин не потрібно вказувати guard бо потрібно щоб всі користувачі могли отримати список новин.

Для додання нової новини потрібно використовувати метод post, а для отримання новини використовувати метод get. Також всю реалізацію доцільно перенести з контролера до сервісу notificationService. Також notificationService підключається до котролера за допомогою dependency injection, що присутній у фреймворці nestJS. Контролер для менеджменту новин гуртожитку має два ендпоінта з назвами add для додавання нової новини, findAll для отримання всіх новин гутожитку і findById для отримання однієї новини, використовуючи id новини. На рисунку 3.13 зображено лістинг коду контролера новини гуртожитка університету.

```
@Controller('')
export class NotificationController {
    constructor(private readonly notificationService: INotificationService) { }

    @Post('notification')
    @UseGuards(BearerTokenAuthGuard)
    @Roles('comendant')
    async add(@Req() request: CustomRequest, @Body() notification: Notification): Promise<string> {
        await this.notificationService.add(notification);
        return 'ok';
    }

    @Get('notification')
    async findAll(): Promise<Notification[]> {
        return await this.notificationService.findAll();
    }

    @Get('notification/:id')
    async findById(@Param('id') id: string): Promise<Notification> {
        return await this.notificationService.findById(id);
    }
}
```

Рисунок 3.13 – Лістинг коду контролера функціоналу новини гуртожитку

Сервіс для менеджменту новин делегує додавання та отримання новин для гуртожитку до інтерфейсу репозиторію, що відповідає за взаємодією з базою даних. Репозиторій для новин гуртожитку використовує MongoDB як базу даних. При створенні об’єкту репозиторію новин підключаються об’єкти notificationModel та notificationMapper. Об’єкт notificationModel відповідає за взаємодію з базою даних та за виконання операцій з базою даних. Об’єкт notificationMaper відповідає за перетворення об’єктів, що зберігаються у базі даних у об’єкти, що використовує застосунок. На рисунку 3.14 зображено лістинг коду репозиторію новин гуртожитку.

```
@Injectable()
export class MongoNotificationRepository implements INotificationRepository {
    constructor(@InjectModel(NotificationDocument.name) private notificationModel: Model<NotificationDocument>,
        private notificationMapper: NotificationMapper) { }

    async findAll(): Promise<Notification[]> {
        return (await this.notificationModel.find().exec()).map(n => this.notificationMapper.mapToModel(n));
    }

    async findById(id: string): Promise<Notification | null> {
        return this.notificationMapper.mapToModel((await this.notificationModel.findById(id).exec()));
    }

    public async add(notification: Notification): Promise<void> {
        const notificationDocument = this.notificationMapper.mapToEntity(notification);
        const notificationModel = new this.notificationModel(notificationDocument);
        await notificationModel.save();
    }
```

Рисунок 3.14 – Лістинг коду репозиторію новин гуртожитку

Однією з найголовніших функціональностей застосунку для підвищення ефективності процесу управління гуртожитку університету є автоматичний розрахунок боргу студента. Розрахунок боргу студента відбувається коли відкривається сторінка проплат студента. Розрахунок боргу студента є лише компонентою для сторінки менеджменту проплат студента. Компонента управління для проплат студента використовує компоненту PaymentCardComponent для відображення списку всіх проплат для студента. Список всі проплат для студента завантажується при завантаженні сторінки або при її перезавантаженні. Також для користувачів, що мають роль student відображається функціонал для добавлення нових проплат. Студенти можуть вибрати файл проплати та зберегти проплату. При завантаженні проплати, перший запит з проплатою піде на розпізнання суми проплати. Після отримання відповіді від сервісу розпізнання проплати, результат розпізнання та картинка проплати відправляється у API сервіс для зберігання проплати. Також у коді компоненти сторінки проплат студента присутні атрибути className, що відповідаються за css, що покращує дизайн інтерфейсу та візуальне сприйняття користувачем застосунку. На рисунку 3.15 зображено лістинг коду компоненти сторінки проплат студента.

```
return (
    <div>
        <div className='blackLine'></div>
        <div className='paymentWrapper'>
            {
                <div>
                    {
                        role === 'student' && componentStrategy === NavigationUrls.STUDENT_PAYMENT &&
                        <div className='paymentLoadWrapper'>
                            <input type='file' onChange={handleChangePaymentCheck} />
                            <button className='save-btn' onClick={savePymentCheck}>Зберегти</button>
                        </div>
                    }
                    <div className='paymentDeptWrapper'>
                        <p>{debtWord}</p>
                        <p>{debtMoney}</p>
                    </div>
                </div>
            }
        </div>
        <div className='payments'>
            {paymentChecksData.map((payment, i) => {
                return (<PaymendCardComponent key={i} props={payment} role={role} />)
            })}
        </div>
    </div>
)
```

Рисунок 3.15 – Лістинг коду сторінки для проплати студента

Взаємодія між фронтендом та бекендом функціоналу відбувається завдяки сервісу боргу. Компонент для розрахування боргу використовує сервіс взаємодії між фронтендом та бекендом для відправки запитів для розрахунку боргу для студента. Сервіс взаємодії для боргу студента робить get запит на backend сервіс для отримання боргу студента. Також для отримання боргу потрібно передати параметр userId для уточнення для якого студента розраховується борг. Також реквест виконується за допомогою сервісу apiRequestService, що разом зі звичайним запитом відправляє хедер Authorization з токеном для авторизації у застосунку. Лістинг коду сервісу взаємодії для розрахунку боргу зображено на рисунку 3.16.

```
class DebtService {
    async getDebtByUserId(userId: any) {
        const result = await apiRequestService.makeRequest(AxiousRequestMethod.get,
            `${process.env.REACT_APP_DIPLOMNA_API_URL_BASE}/debt/${userId}`);
        return result;
    }
}
const debtService = new DebtService();
export default debtService;
```

Рисунок 3.16 – Лістинг коду сервісу взаємодії для розрахунку боргу

Функціонал для розрахунку боргу студента потребує встановлення ціни за місяці та завантаження студентом проплат за проживання у гуртожитку.

Розраховувати борг для студента можуть всі користувачі системи, що увійшли у систему тому присутній guard з користувачами comendant, student, dean, passportHolder. Також існує перевірка, якщо користувач зробить реквест на ендпоінт getDebt з роллю student, то цей користувач повинен відправити параметр userId для себе самого, оскільки студенти не повинні мати можливості дізнаватися борг для інших користувачів. Також реалізація розрахунку боргу для студента перенесена з контролеру до сервісу debtService, що підставляється фреймворком nestJS за допомогою dependency injection. Також у анотації @Get поставлено url ‘debt/:userId’, бо ендпоінт очікує отримати userId. Лістинг коду контролера для розрахунку боргу студента зображено на рисунку 3.17.

```
@Controller('')
export class DebtController {
    constructor(private readonly debtService: IDebtService) { }
    @Get('debt/:userId')
    @UseGuards(BearerTokenAuthGuard)
    @Roles('comendant', 'student', 'dean', 'passportHolder')
    async getDebt(@Req() request: CustomRequest, @Param('userId') userId: string): Promise<Debt> {
        if (request.role === 'student' && request.userId !== userId) {
            throw new ForbiddenException('Forbidden');
        }
        return await this.debtService.calculateDebt(userId)
    }
}
```

Рисунок 3.17 – Лістинг коду контролера розрахунку боргу студента

Розрахунок боргу студента передбачає використання сервісів для отримання місяців, що прожив студент у гуртожитку та цін до них, а також сервісу для проплат. На рисунку 3.18 зображено код для розрахунку боргу студента.

```
public async calculateDebt(userId: string): Promise<Debt> {
    const rentHistories: RentHistory[] = await this.rentHistoryService.getRentHistoryByUserId(userId);
    const rentHisotryMonths = this.getMonthsForRentHistories(rentHistories);
    const monthPrice: number = await this.monthPriceService.getPriceByMonths(rentHisotryMonths);
    const paymentValues = await this.paymentService.findPaymentValuesByUserId(userId);
    const paymentSum = this.getPaymentSum(paymentValues);
    return {
        value: monthPrice - paymentSum,
        userId: userId
    };
}

private getPaymentSum(paymentValues: number[]): number {
    let paymentValueSum: number = 0;
    paymentValues.forEach(paymentValue => {
        paymentValueSum += paymentValue;
    });
    return paymentValueSum;
}
```

Рисунок 3.18 – Лістинг коду розрахунку боргу студента

Для розрахунку боргу спочатку потрібно отримати історію проживання студента у гуртожитку шляхом отримання її через сервіс rentHistoryService, для цього потрібно передати userId. Потім потрібно отримати місяці, що проживав студент у гуртожитку використовуючи історію проживання. Для отримання суми всіх цін за місяць потрібно звернутись до сервісу monthPriceService та передати місяці, що прожив студент у гуртожитку. Для того, щоб отримати проплати потрібно звернутись до сервісу findPaymentValuesByUserId та передати userId студента для якого розраховується боргу. Після цього сумуються всі затверджені проплати та записуються у змінну paymentSum. У результаті сума визначається різницею сум всіх цін за місяць, що прожив студент у гуртожитку та сум підтверджених проплат студента, а саме monthPrice мінус paymentSum.

Також важливою функціональністю є розпізнання суми проплати студента. Студент завантажує проплату на сервер, де розпізнається картинка і студенту повертається інформація про суму проплати, Id транзації та рахунок отримувача. На рисунку 3.19 зображено лістинг коду сервісу як FrontedService взаємодіє з сервісом розпізнання зображення.

```
class ImageRecognitionsService {
    public async recognizePaymentCheck(paymentCheckImage: any) {
        const formData = new FormData();
        formData.append('file', paymentCheckImage, 'file');
        const url = `${process.env.REACT_APP_IMAGE_RECOGNITION_BASE_URL}/recognize/paymentCheck`;
        return await apiRequestService.makeRequest(AxiousRequestMethod.post, url, formData);
    }
}

const imageRecognitionsService = new ImageRecognitionsService();
export default imageRecognitionsService;
```

Рисунок 3.19 – Лістинг коду взаємодії фронтенду та бекенду розпізнання суми проплати

Для розпізнання суми проплати зображення потрібно, спочатку отримати від користувача картинку проплати. Отримання проплати відбувається за

допомогою Http запиту, а потім після отримання картинки викликається сервіс реалізації розпізнання суми. Сервісом розпізнання суми є paymentCheckRecognitionService, що підключається за допомогою dependency injection, що працює за допомогою ASP NET фреймворк. Також для класу контролера присутній атрибут Authorize з ролями student, admin, comendant, dean, passportHolder, що означає що всі користувачі, що увійшли у застосунок, мають доступ до контролеру розпізнання проплати. Лістинг коду контролера для розпізнання суми проплати студента зображено на рисунку 3.20.

```
[Authorize(Policy = LocalApi.PolicyName, Roles = "student,admin,comendant,dean,passportHolder")]
Ссылок: 1
public class ImageRecognitionController : ApiController
{
    private readonly IPaymentCheckRecognitionService _paymentCheckRecognitionService;
    Ссылок: 0
    public ImageRecognitionController(IPaymentCheckRecognitionService paymentCheckRecognitionService) {
        _paymentCheckRecognitionService = paymentCheckRecognitionService;
    }

    [HttpPost]
    [Route("/recognize/paymentCheck")]
    Ссылок: 0
    public CheckPaymentInfo RecognizePaymentCheck()
    {
        return _paymentCheckRecognitionService.RecognizeCheckPaymentInfo(GetImageFromRequestBytes());
    }
}
```

Рисунок 3.20 – Лістинг коду контролера розпізнання проплати

Реалізація розпізнання суми проплати студента відбувається з використанням різних стратегій для забезпечення розпізнання різних проплат для різних банків. Кожна стратегія розпізнає потрібну інформацію з проплати, після розпізнання для всіх стратегій, та стратегія яка розпізнала найбільше потрібної інформації буде повертати користувачу розпізнану інформацію. Розпізнання тексту зображень відбувається з використанням бібліотеки Tesseract. Розпізнана інформація повертається користувачу у вигляді об’єкта з розпізнаною сумою проплати, ідентифікатора транзакції та рахунку отримувача. Лістинг коду реалізації розпізнання суми проплати зображено на рисунку 3.21.

```
public class PaymentCheckRecognitionService : IPaymentCheckRecognitionService
{
    private List<IPaymentCheckStrategy> paymentCheckStrategies = new List<IPaymentCheckStrategy>()
    {
        new PrivatBankDesktopPaymentCheckStrategy(),
        new MonoBankDesktopPaymentCheckStrategy(),
        new PrivatBankPaperPaymentCheckRegognitionStrategy()
    };

    Ссылок: 2
    public CheckPaymentInfo RecognizeCheckPaymentInfo(byte[] image)
    {
        CheckPaymentInfo payemtnInfoResult = null;
        int maxRecognitions = -1;
        foreach (var recognitionStartegy in paymentCheckStrategies)
        {
            int recognizeResult = recognitionStartegy.Recognize(image, out CheckPaymentInfo payemtnInfo);
            if (recognizeResult > maxRecognitions)
            {
                payemtnInfoResult = payemtnInfo;
                maxRecognitions = recognizeResult;
            }
        }
        return payemtnInfoResult;

    }
}
```

Рисунок 3.21 – Лістинг коду реалізації розпізнання суми проплати

Також на даний момент присутні стратегії розпізнання для десктопної версії квитанції «ПриватБанку», паперової версії квитанції «ПриватБанку» та десктопної версії квитанції «MonoBank». Було вибрано за приклад розпізнання суми проплати для десктопної квитанції «ПриватБанку». Метод розпізнання проплати отримує ініціалізований об’єкт TesseractEngine, що використовується для розпізнання тексту для зображень. Також передається зображення проплати у змінній img. Першим кроком буде створення області для розпізнання тексту на зображенні. Потім, використовуючи TesseractEngine, розпізнати текст та записати розпізнаний текст у змінну recognizedText. Потім, використовуючи Regex, знайти суму проплати у тексті, якщо кількість пар є одною, то повернути розпізнану інформацію, інакше повернути мінус одиницю. Також якщо під час розпізнання виникне помилка, то потрібно буде повернути також мінус одиницю. Також тип, що повертає функція розпізнання є double, що одначає що перед відправкою потрібно парсити результат у тип double. На рисунку 3.22 зображено лістинг коду стратегії розпізнання суми проплати для десктопної квитанції «ПриватБанку».

```
private double GetCheckPaymentPrice(TesseractEngine engine, Pix img)
{
    try
    {
        Rect roi = new Rect(img.Width * 15 / 20, 1, img.Width * 5 / 20, img.Height - 2);
        using (var page = engine.Process(img, roi))
        {
            // Get the recognized text
            string recognizedText = page.GetText();
            string pattern = @"(?<!\d)(\d+\.\d+)(?!\d)";
            Regex regex = new Regex(pattern);
            var matches = regex.Matches(recognizedText);
            if (matches.Count == 1)
            {
                return double.Parse(matches[0].Value, CultureInfo.InvariantCulture);
            }
            else
            {
                return -1;
            }
        }
    }
    catch (Exception exc)
    {
        return -1;
    }
}
```

Рисунок 3.22 – Лістинг коду розпізнання суми проплати студента для десктопної стратегії «ПриватБанку»

Найважливішою функціональністю застосунку для підвищення ефективності процесу управління гуртожитку є можливість обліку інформації про студентів. Комендант має можливість отримати список всіх студентів, що колись проживали або будуть проживати у гуртожитку та змінювати інформацію в їх профілі про них. За допомогою списку студентів комендант або керуючий персонал зможе отримати доступ до профілю студента, проплат студента, заяви на проживання студента або до історії проживання студента у гуртожитку. Ініціалізація списку користувачів відбувається при першому заході на сайт або при перезавантаженні сторінки. Також компонент керування менеджменту студентів використовує компонент UserProfileListCard, ща має у собі кнопки керування для переходу на профіль користувача, борг користувача, заяву на проживання користувача та історію проживання користувача у гуртожитку університету. Лістинг коду компоненту менеджменту студентів зображено на рисунку 3.23.

```
export default function UserProfileList(props: any) {

    const [users, setUsers] = useState([]);

    const initializeData = async () => {
        usersService.getUsersList().then((result) => {
            console.log(result.data);
            setUsers(result.data);
        });
    };

    useEffect(() => {
        initializeData();
    }, []);


    return (
        <div>
            {users.map((user: any, i) => {
                // Return the element. Also pass key
                return (<UserProfileListCard key={i} userId={user.id} userName={user.userName} />)
            })}
        </div>
    )
}
```

Рисунок 3.23 – Лістинг коду компоненти менеджменту студентів

Також керуючий персонал може переглядати профілі будь-яких користувачів, а студент може переглядати та робити зміни лише у свому профілі. Студент може вказувати у профілі своє ім’я, прізвище, ім’я по батькові, пошту, номер паспорту та завантажувати картинку свого обличчя та картинку свого підпису. Профілі керуючого персоналу також використовують цей компонент, але не містять у собі поля номеру паспорту, оскільки присутня перевірка, що це поле відображається лише для користувачів з роллю студента. На рисунку 3.24 зображено лістингу коду інтерфейсу профілю користувача.

```
<div className='prifileWrapper'>
    <div className='leftBlock'>
        <div className='controll-input'>
            <p>Ім'я</p>
            <input type="text" value={firstName} onChange={handleChangeFirstName} />
        </div>

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

        <div className='controll-input'>
            <p>По Батькові</p>
            <input type="text" value={thirdName} onChange={handleChangeThirdName} />
        </div>
```

Рисунок 3.24 – Лістинг коду інтерфейсу профілю користувача

Для забезпечення можливості зберігання, редагування та отримання профілів користувачів було реалізовано сервіс менеджменту профілів. Сервіс

менеджменту профілів делегує зберігання, отримання та редагування до репозиторію профілів, що відповідає за взаємодію з базою даних. Реалізація інтерфейсу репозиторію підключається автоматично за допомогою фреймворку nestJS. На рисунку 3.25 зображена реалізація сервісу менеджменту інформації користувача.

```
@Injectable()
export class DefaultUserProfileService implements IUserProfileService {

    constructor(private userProfileRepository: IUserProfileRepository) {

    }

    async findAll(): Promise<UserProfile[]> {
        return await this.userProfileRepository.findAll();
    }

    /*async add(notification: Notification): Promise<void> {
        await this.notificationRepository.add(notification);
    }*/

    async findById(id: string): Promise<UserProfile | null> {
        return await this.userProfileRepository.findById(id);
    }

    async updateImage(id: string, imageDto: any): Promise<void> {
        await this.userProfileRepository.updateImage(id, imageDto);
    }

    async update(id: string, userProfile: UserProfile): Promise<void> {
        await this.userProfileRepository.update(id, userProfile);
    }

}
```

Рисунок 3.25 – Лістинг коду сервісу менеджменту інформації користувача

Також допоміжною функціональністю застосунку для підвищення ефективності процесу управління гуртожитком є генерація документів з використанням поточної інформації користувачів та їх підписів. Коли користувач робить запит на генерацію документа, сервіс генерації документів робить запит на ApiService та отримує всю необхідну інформацію для генерації документа. Генерація документа відбувається шляхом заміни у документі певних ключових слів на картинки та потрібну інформацію. Тобто сервіс вибирає потрібний word шаблон, замінює інформацію та передає користувачу word документ. Заміна інформації в документі відбувається за допомогою бібліотеки DocX. Спочатку шаблон копіюється в тимчасове місце, потім скопійований

шаблон змінюється шляхом заміни інформації та вставленні потрібних картинок. Після цього змінений тимчасовий файл конвертується в масив байтів та передається користувачу. Після відправки користувачу потрібного документу тимчасовий файл видаляється. Лістинг коду генерації документу зображено на рисунку 3.26.

```
public MemoryStream Word(string applicationForSettlementId)
{
    ApplicationForSettlementDto applicationForSettlement = _applicationForSettlementService.
        GetApplicationForSettlement(applicationForSettlementId);
    string comendantUserId = GetComendantUserId(applicationForSettlement);
    string deanUserId = GetDeanUserId(applicationForSettlement);
    string studentUserId = GetStudentUserId(applicationForSettlement);
    UserProfileDto comendantProfile = _userProfileService.GetProfile(comendantUserId);
    UserProfileDto deanProfile = _userProfileService.GetProfile(deanUserId);
    UserProfileDto studentProfile = _userProfileService.GetProfile(studentUserId);
    string tempFilePath = Path.GetTempFileName();
    tempFilePath = Path.ChangeExtension(tempFilePath, "docx");
    System.IO.File.Copy($@".{Path.DirectorySeparatorChar}WordSamples
    {Path.DirectorySeparatorChar}ApplicationForSettlement.docx",
        tempFilePath, true);
    var doc = DocX.Load(tempFilePath);
    SetInformationForWord(doc, studentProfile, deanProfile, comendantProfile, applicationForSettlement);
    GC.Collect();
    GC.WaitForPendingFinalizers();
    doc.SaveAs(tempFilePath);
    return WordGeneratorUtils.GetMemoryStreamWord(tempFilePath);
}
```

Рисунок 3.26 – Лістинг коду генерації документу

Для генерації документу спочатку отримується інформація про ордер та записується у змінну applicationForSettlement. З ордера можна отримати Id коменданта, що підписав ордер, Id декана, що також підписав ордер, Id студента, що створив ордер. Потім отримується інформація про профілі користувачів для коменданта, декана та студента. Потім створюється назва для тимчасового файлу, що генерується за допомогою .NET методі GetTempFileName. Потім добавляється розширення docx до тимчасового файлу. Потім копіюється шаблон з місця зберігання до тимчасового файлу, тимчасовий файл відкривається за допомогою бібліотеки DocX. У методі SetInformationForWord замінюється інформація у шаблоні та підставляються картинки по ключовим словам у шаблоні. Після виконання функцій заміни тимчасовий word файл зберігається та перетворюється у масив байтів, що передається користувачу, тимчасовий файл видаляється. На рисунку 3.27 зображено лістинг коду для заміни інформації у

шаблоні по ключовим словам.

```
private void SetInformationForWord(DocX doc, UserProfileDto studentProfile, UserProfileDto deanProfile
    , UserProfileDto comendantProfile, ApplicationForSettlementDto applicationForSettlement)
{
    XDocWordGeneratorUtils.ReplaceText(doc, "${student}",
        $"{studentProfile.SecondName} {studentProfile.FirstName} {studentProfile.ThirdName}");
    XDocWordGeneratorUtils.ReplaceText(doc, "${group}", studentProfile.Group);
    XDocWordGeneratorUtils.ReplaceText(doc, "${faculty}", studentProfile.Faculty);
    XDocWordGeneratorUtils.ReplaceText(doc, "${telephone}", studentProfile.Phone);
    string date = applicationForSettlement.Date.Length > 10
        ? applicationForSettlement.Date.Substring(0, 10) : applicationForSettlement.Date;
    XDocWordGeneratorUtils.ReplaceText(doc, "${date}", date);
    XDocWordGeneratorUtils.ReplaceText(doc, "${startYear}", applicationForSettlement.StartYear.ToString());
    XDocWordGeneratorUtils.ReplaceText(doc, "${endYear}", applicationForSettlement.EndYear.ToString());
    if (studentProfile.SignImage != null)
    {
        XDocWordGeneratorUtils.ReplaceImage(doc, "image1.jpeg", Convert.FromBase64String(studentProfile.SignImage));
    }
    XDocWordGeneratorUtils.ReplaceText(doc,
        "${dean}", $"{deanProfile.SecondName} {deanProfile.FirstName} {deanProfile.ThirdName}");
    if (deanProfile.SignImage != null)
    {
        XDocWordGeneratorUtils.ReplaceImage(doc, "image2.jpeg", Convert.FromBase64String(deanProfile.SignImage));
    }
}
```

Рисунок 3.27 – Лістинг коду заміни інформації у word шаблоні

Отже, у підрозділі було проведено опис коду функціоналу розроблюваного програмного застосунку для підвищення ефективності процесу управління гуртожитку університету. Повний лістинг наведено у додатку В.

3.5 Висновки

Отже, у третьому розділі було зроблено варіантний аналіз мов програмування та було вибрано мову програмування TypeScript як основну мову програмування. Також було проведено аналіз середовищ розробки у результаті чого було вибрано Visual Studio Code, як середовище розробки, що буде використовуватись під час розробки застосунку. Було вибрано MongoDB як основну базу даних, що буде використовуватися застосунком. Також було вибрано React та описано розробку інтерфейсу за допомогою React бібліотеки.

Було визначено основний функціонал розроблюваного програмного застосунку та було проведено опис коду реалізації застосунку.

4 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ПРОДУКТУ

4.1 Аналіз методів тестування програмного забезпечення

Тестування програмного забезпечення це найважливіший етап розробки, оскільки під час цього етапу визначається робото здатність продукту та відповідність його функціональним вимогам. Під час цього етапу можна визначити помилки та вирішити їх. Існує велика кількість видів тестування програмного забезпечення, що відрізняються за знанням системи, степенем автоматизації, затраченим часом та іншим.

Розглянемо види тестування за знанням системи: тестування білої скриньки, тестування сірої скриньки, тестування чорної скриньки.

Тестування білої скриньки – це метод тестування програмного продукту, при якому тестувальник має повний доступ до внутрішньої структури і коду програми [25]. Основною метою цього виду тестування є виявлення помилок, що можуть виникнути під час виконання програми, на рівні її коду та архітектури. Тестування «білої скриньки» передбачає ретельний аналіз коду програми, використання тестових даних для виклику різних функцій та методів, і визначення, чи працює програма належним чином і чи відповідає специфікаціям. Цей вид тестування дозволяє виявити проблеми, пов'язані з логікою програми, покриттям коду та іншими аспектами, які можуть залишитися невидимими при інших методах тестування. Тестувальники «білої скриньки» можуть використовувати різні техніки, такі як тестування шляху виконання, аналіз підпрограм та інші, для виявлення помилок. Цей підхід є важливим для забезпечення якості програмного продукту та забезпечення його надійності. До переваг тестування «білої скриньки» можна віднести ретельність проведення тестування та можливість проведення тестування на різних стадіях розробки ПЗ. Важливим недоліком є те, що тестувальник повинен повинен розуміти внутрішній устрій програми.

Тестування сірої скриньки – це метод тестування програмного забезпечення, який поєднує в собі як елементи білого, так і чорного ящика [26]. У

«сірій сриньці» тестувальник має деяку обмежену інформацію про внутрішню структуру програми. Цей підхід дозволяє здійснювати тестування більш комплексних систем, враховуючи як функціональні аспекти, так і структурні особливості. Тестувальник отримує можливість перевірити, як система взаємодіє зі своїм оточенням і з зовнішніми даними, одночасно аналізуючи її внутрішню логіку. Тестування за допомогою «сірої скриньки» може бути особливо корисним для виявлення вразливостей, які можуть бути використані для зловмисних дій, а також для оптимізації продуктивності системи та виявлення помилок, які можуть призвести до відмови програми, допомагає покращити загальну якість програмного забезпечення та забезпечити його надійність. Недоліком є те що, тестування за допомогою «сірої скриньки» вимагає спеціалізованих навичок та знань, а також доступу до обмеженої інформації про програму.

Тестування чорної скриньки – це метод тестування програмного забезпечення, який спрямований на перевірку функціональності програми без необхідності знання її внутрішньої структури [27]. У процесі такого тестування тестер не має доступу до вихідних кодів програми та внутрішніх деталей її реалізації. Замість цього, він взаємодіє з програмою як інший користувач, вводячи дані та спостерігаючи за реакцією системи. Основні переваги тестування «чорної скриньки» полягають у тому, що вони дозволяють оцінити програму з позиції кінцевого користувача та виявити проблеми, які можуть залишитися непоміченими під час тестування «білої скриньки» або інших методів, що базуються на знанні внутрішньої структури програми. Таким чином, тестування «чорної скриньки» є важливою складовою процесу валідації програмного забезпечення і сприяє підвищенню його якості та надійності. Недоліком є те що, без знання внутрішньої структури програми важко складати тест-кейси.

Отже, у результаті розгляду усіх видів тестування за знанням системи було вирішено, що найкращим рішенням буде використовувати методику «чорної скриньки» з попереднім створенням набору тест-кейсів. Вид тестування «чорної скриньки» дозволить виявляти помилки у роботі застосунку з точки зору користувача.

4.2 Тестування розробленого програмного продукту

Тестування програмного забезпечення методом «чорної скриньки» відбувається за допомогою тест-кейсів [28]. Текст-кейсом є розроблений алгоритм тестування функцій програмного забезпечення. Основними функціями застосунку для підвищення ефективності процесу управління гуртожитку університету є: функція розрахунку боргу студента, функція розпізнання проплати студента, можливість генерації документів з поточною інформацією та можливість обліку інформації про студента. Також допоміжною функцією підвищення ефективності управління гуртожитку є функція публікування новин для гуртожитку. У функціонал розрахунку боргу студента входить функціонал встановлення боргу за проживання у гуртожитку за місць, та функція обліку проплат студента.

Для перевірки функціоналу застосунку було розроблено тест-кейси:

Тест-кейс №1 – Встановлення ціни за місяць проживання у гуртожитку:

1. Виконати вхід до системи під користувачем для керуючого персоналу.
2. Ввійти у вкладку «Ціна проживання за місяць».
3. Ввести у поле «Місяць» місяць.
4. Ввести у поле «Ціна за місяць» ціну проживання за місяць у гуртожитку.
5. Натиснути кнопку «Зберегти».

Очікуваним результатом виконання тест-кейсу є створення нового запису ціни проживання за місяць. Також місяць проживання та ціна проживання у новому записі повинні відповідати введеним даним. Новий запис повинен бути доступним до перегляду для керуючого персоналу.

Введемо у поле «Місяць» місяць проживання «2023-10» також введемо у поле «Ціна проживанняза місяць» 960.5 грн і натиснемо кнопку «Зберегти». Результат виконання тест-кейсу наведено на рисунку 4.1.

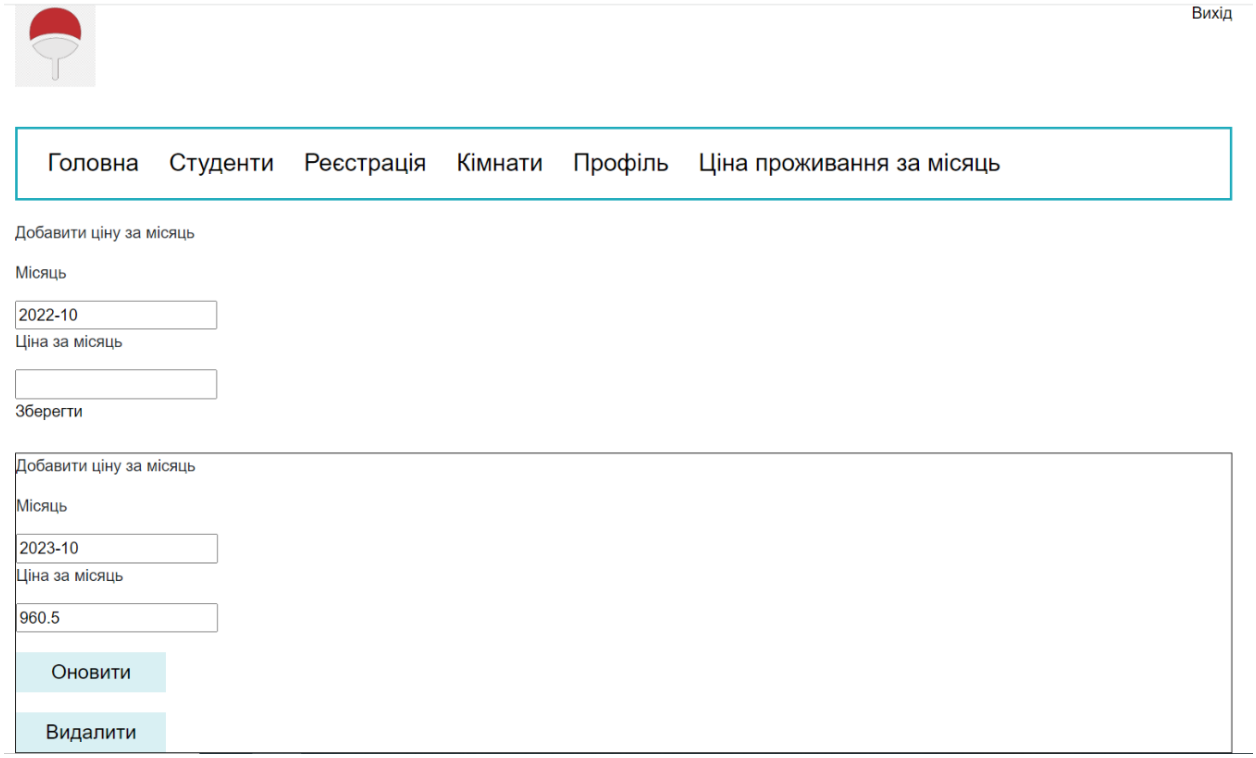


Рисунок 4.1 – Результат виконання тест-кейсу №1

В результаті тест-кейсу №1 було додано запис ціни за місяць «2023-10» з ціною 960.5 грн, що відповідає очікуваному результату. Також новий запис доступний для перегляду для керуючого персоналу. Тому тест-кейс №1 успішно пройдено.

Тест-кейс №2 – Поселення студента у кімнату:

1. Ввійти у систему під користувачем керуючого персоналу.
2. Натиснути на вкладку «Студенти».
3. Натиснути на кнопку «Проживання».
4. Зановнити поле «roomId».
5. Натиснути кнопку «Поселити».

Очікуваним результатом виконання є додання запису про проживання студента. Початком проживання студента повинен бути поточний місяць. Також кімната у новому записі повинна відповідати введеній кімнаті. Новий запис повинен добавлятись в кінець списку.

Введемо існуючу кімнату у гуртожитку у поле «roomId» та натиснемо кнопку «Поселити». Результат виконання тест-кейсу наведено на рисунку 4.2.

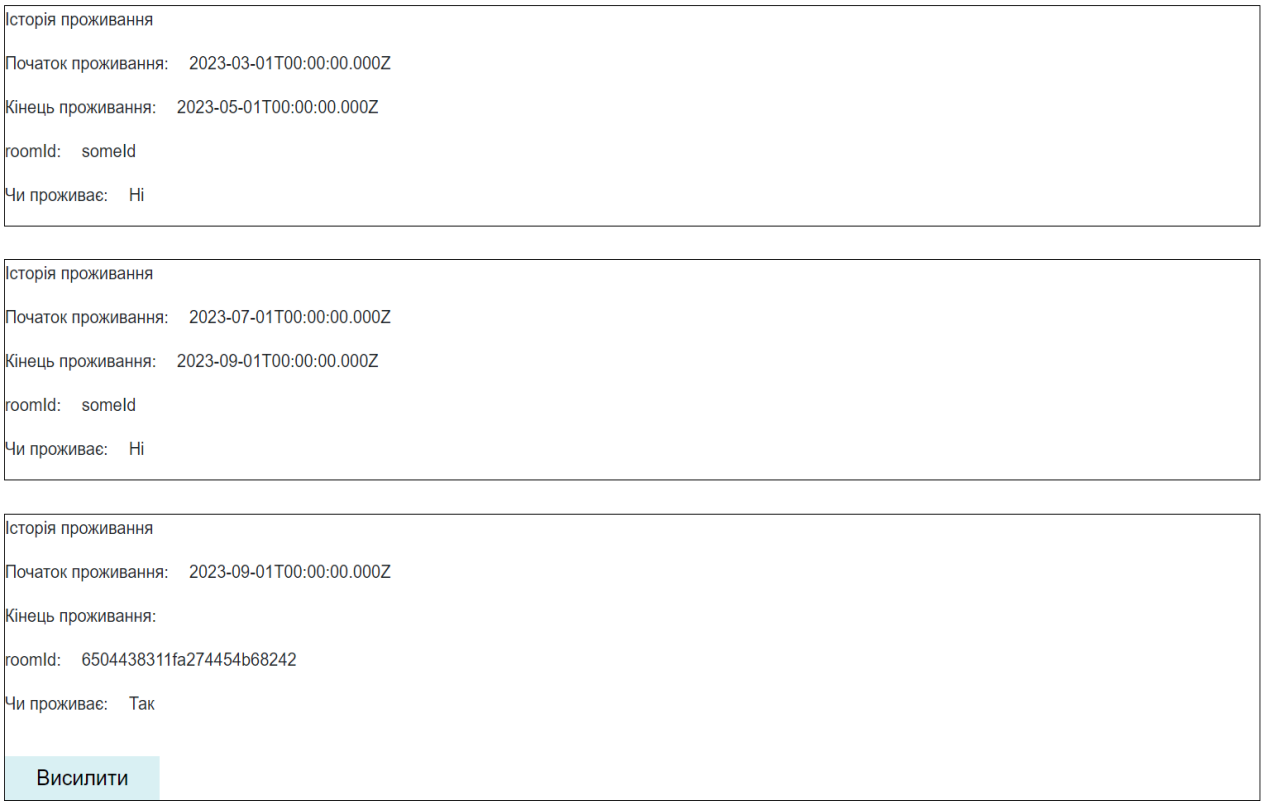


Рисунок 4.2 – Результат виконання тест-кейсу №2

У результаті виконання тест-кейсу №2 було додано запис про проживання студента у гуртожитку, початком проживання у гуртожитку є поточний місяць, а кімната в яку було поселено студента відповідає введеній. «Кінець проживання» є пустим, оскільки студента не виселлено. Також новий запис додано в кінець списку. Результат тест-кейсу №2 відповідає очікуваному результату, тому тест-кейс №2 успішно пройдено.

Тест-кейс №3 – Внесення проплати студента та розпізнання суми проплати:

1. Вхід у систему під користувачем студента.
2. Натиснути кнопку «Проплата».
3. Обрати файл для завантаження.
4. Натиснути кнопку «Зберегти».

Очікуваним результатом є отримання запису з проплатою, розпізнаною ціною проплати, розпізнаним кінцевим рахунком та Id проплати. Результат виконання тест-кейсу №3 наведено на рисунку 4.3.

Рисунок 4.3 – Результат виконання тест-кейсу №3

У результаті виконання тест кейсу було отримано запис з квитанцією проплати, розпізнаними: сумою, кінцевим рахунком та ідентифікатором проплати. Розпізнана ціна, розпізнаний кінцевий рахунок отримувача та розпізнаний ідентифікатор чеку повністю відповідають внесеній проплаті. Результат тест-кейсу №3 відповідає очікуваному результату, тому тест-кейс №3 успішно пройдено.

Тест-кейс №4 – Розрахунок боргу студента:

1. Вхід у систему під користувачом керуючого персоналу.
2. Натиснути кнопку «Студенти».
3. Натиснути кнопку «Проплата».
4. Підтвердити проплату студента.
5. Вийти зі системи.
6. Вхід у систему під користувачем студента.
7. Натиснути кнопку «Проплата».

Очікуваним результатом є коректний розрахунок боргу студента. При переплаті перед боргом повинно стояти слово «Переплата». «Борг», якщо немає переплати. Результат виконання тест-кейсу наведено на рисунку 4.4.

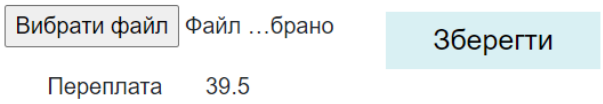


Рисунок 4.4 – Результат виконання тест-кейсу №4

У результаті виконання тест-кейсу було розраховано борг студента. Студент проживав у гуртожитку 2023-10 місяць. Ціна проживання за місяць 2023-10 була встановлена як 960.5 грн, розпізнана проплата студента була 1000 грн, розрахований борг студента це переплата у 39.5 грн, що є коректним боргом. Перед боргом стоїть слово «Переплата». Тест-кейс №4 успішно пройдено.

Тест-кейс №5 – Заповнення інформації про студента:

1. Увійти у систему під користувачем студента.
2. Натиснути кнопку «Профіль».
3. Заповнити поле «Ім’я».
4. Заповнити поле «Прізвище».
5. Заповнити поле «По Батькові».
6. Заповнити поле «Телефон».
7. Заповнити поле «Пошта».
8. Заповнити поле «Номер паспорта»
9. Заповнити поле «Група».
10. Заповнити поле «Факультет».
11. Заповнити поле «Дата отримання паспорту».
12. Заповнити поле «Дата народження».
13. Заповнити поле «Курс».
14. Завантажити картинку профілю.
15. Завантажити картинку підпису.
16. Натиснути кнопку «Зберегти».

Очікуваним результатом є збереження інформації про студента та отримання необхідної інформації з серверу після перезавантаження сторінки. Результат виконання тест-кейсу зображено на рисунку 4.5.

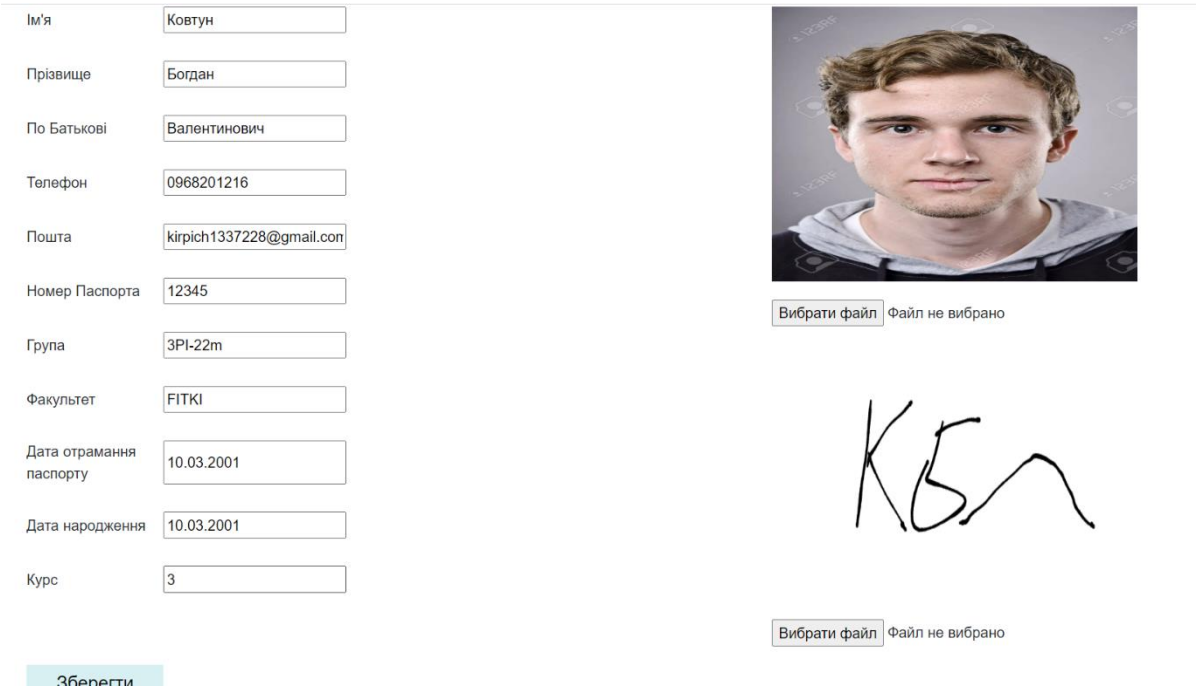


Рисунок 4.5 – Результат виконання тест-кейсу №5

У результаті виконання тест-кейсу була збережена інформація про студента, що була введена, інформація відповідає очікуваному результату. Інформація присутня після перезавантаження сторінки. Тест-кейс №5 пройдено.

Тест-кейс №6 – Добавлення новини про гуртожиток:

1. Увійти у систему під користувачем керуючого персоналу.
2. Натиснути кнопку «Головна».
3. Заповнити поле «Заголовок».
4. Заповнити поле «Повідомлення».
5. Заповнити поле «Автор».
6. Натиснути кнопку «Розмістити».

Очікуваним результатом виконання тест-кейсу буде добавлення запису з заголовком, повідомленням і автором до новин гуртожитку. Введена інформація у полях повинна відповідати інформації в записі. Також дата у записі повинна бути поточною датою та містити день, місяць та рік створення новини. Нова новина повинна також бути додана на перше місце списку новин.

Заповнимо поля необхідною інформацією та натиснем кнопку «Розмістити». Результат виконання тест-кейсу наведено на рисунку 4.6.

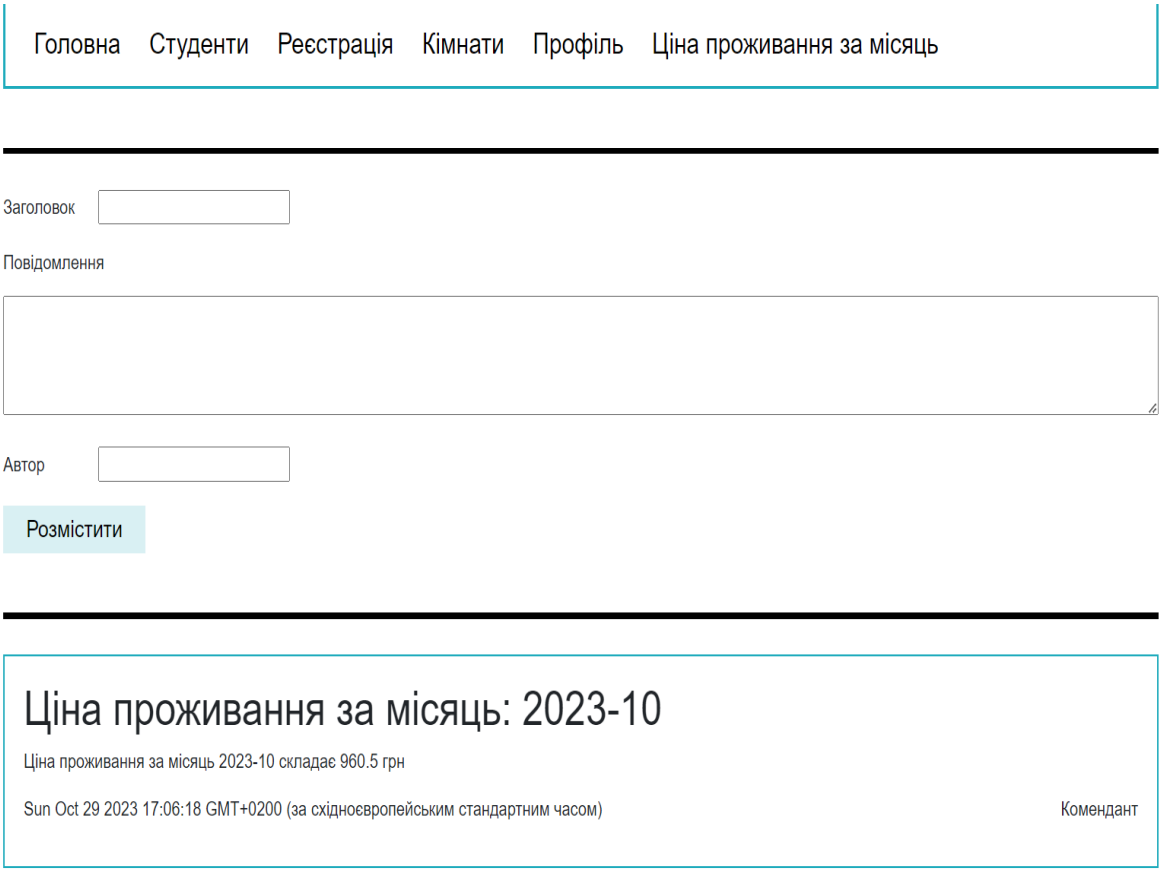


Рисунок 4.6 – Результат виконання тест-кейсу №6

У результаті виконання тест-кейсу було додано запис навини, заголовок новини, основна частина та користувач, що додав новину повністю відповідають введеним даним, що відповідає очікуваному результату. Також дата публікації новини є поточним часом та має день, місяць і рік. Новина була додана на перше місце списку новин. Тест-кейс №6 успішно пройдено.

Тест-кейс №7 – Генерація документу на поселення:

1. Увійти у систему під користувачем студента.
2. Натиснути кнопку «Заява на проживання».
3. Натиснути кнопку «Згенерувати заяву».

Очікуваним результатом є згенерований word файл, що буде мати у собі підписи студента та коменданта та поточну інформацію про студента. Файл повинен відправлятися користувачу при натисненні кнопки «Згенерувати заяву». Результат виконання тест-кейсу наведено на рисунку 4.7.

Студента Богдан Ковтун Валентинович

Групи 3PI-22m

Факультету FITKI

Номер телефону 0968201216

Заява

Прошу надати мені ліжко-місце на 2022 - 2023 н.р. в гуртожитку. Зобов'язуюсь дотримуватися правил внутрішнього розпорядку, ознайомитися із «Положенням про користування гуртожитком вінницького національного технічного університету» та виконувати умови (вимоги) договору (угоди).

29.10.23 (Дата)

(Підпис)

Заступник декана факультету

Дек Д.Д. 29.10.2023

П.І.П. Дата Підпис

Зав. гуртожитком №_5

Коменд К.К 29.10.2023

П.І.П. Дата Підпис

Рисунок 4.7 – Результат виконання тест-кейсу №7

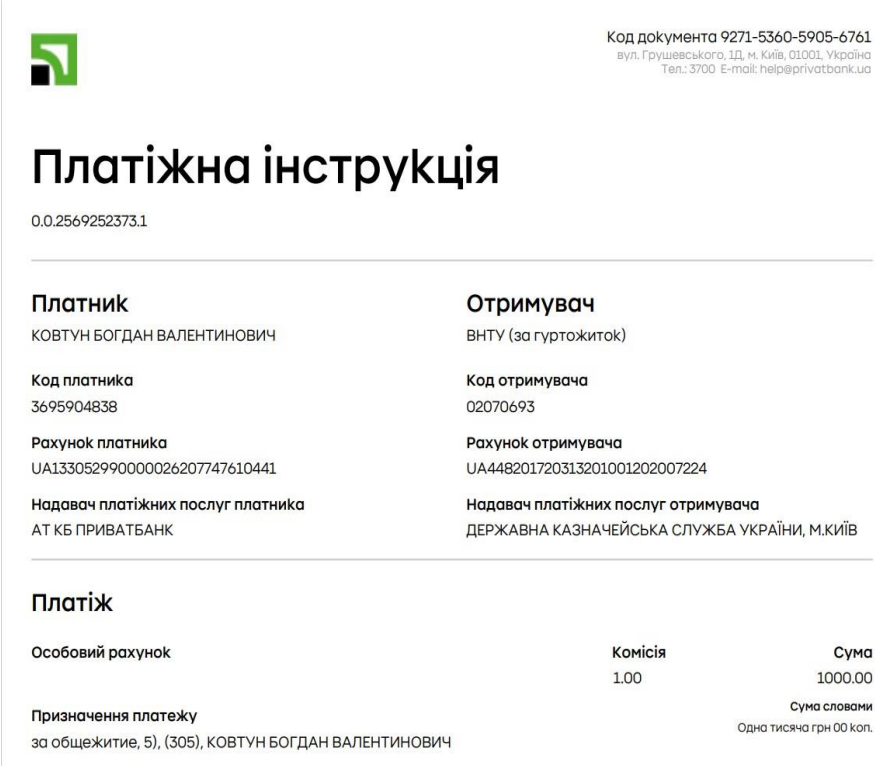
У результаті виконання тест-кейсу було згенеровано word документ на поселення зі всіма необхідними підписами та потрібною інформацією, що відповідає очікуваному результату, тому тест-кейс №7 успішно пройдено.

Отже, у поточному підрозділі було розроблено сім тест-кейсів та протестовано функціонал розроблюваного програмного забезпечення. За результатами тестування, розроблюваний програний продукт підтвердив свою працездатність.

4.3 Експериментальні дослідження методу розпізнання суми проплати, що завантажується студентом

Крім перевірки правильності роботи програмного модуля для розпізнавання суми проплати студента, важливо також перевірити розроблений метод, що використовує даний модуль. Найважливішою характеристикою методу є можливість розпізнавати потрібний текст для різних типів зображень шляхом розпізнання не всього тексту зображення, а лише його певної частини.

Для оцінки правильності підходу розроблюваного методу розпізнання потрібної інформації з зображення проплати доцільно порівняти його з звичайним методом розпізнання інформації з проплати шляхом повного розпізнання тексту для всього зображення. Квитанцію для порівняння зображено на рисунку 4.8.



Код документа 9271-5360-5905-6761
вул. Грушевського, 1Д, м. Київ, 01001, Україна
Тел.: 3700 E-mail: help@privatbank.ua

Платіжна інструкція

0.0.2569252373.1

Платник
КОВТУН БОГДАН ВАЛЕНТИНОВИЧ

Код платника
3695904838

Рахунок платника
UA133052990000026207747610441

Надавач платіжних послуг платника
АТ КБ ПРИВАТБАНК

Отримувач
ВНТУ (за гуртожиток)

Код отримувача
02070693

Рахунок отримувача
UA448201720313201001202007224

Надавач платіжних послуг отримувача
ДЕРЖАВНА КАЗНАЧЕЙСЬКА СЛУЖБА УКРАЇНИ, М.КИЇВ

Платіж

Особовий рахунок

Комісія
1.00

Сума
1000.00

Сума словами
Одна тисяча грн 00 коп.

Призначення платежу
за общежитие, 5), (305), КОВТУН БОГДАН ВАЛЕНТИНОВИЧ

Рисунок 4.8 – Квитанція для порівняння

Результат розпізнання тексту на квитанції шляхом звичайного методу зображено на рисунку 4.9.

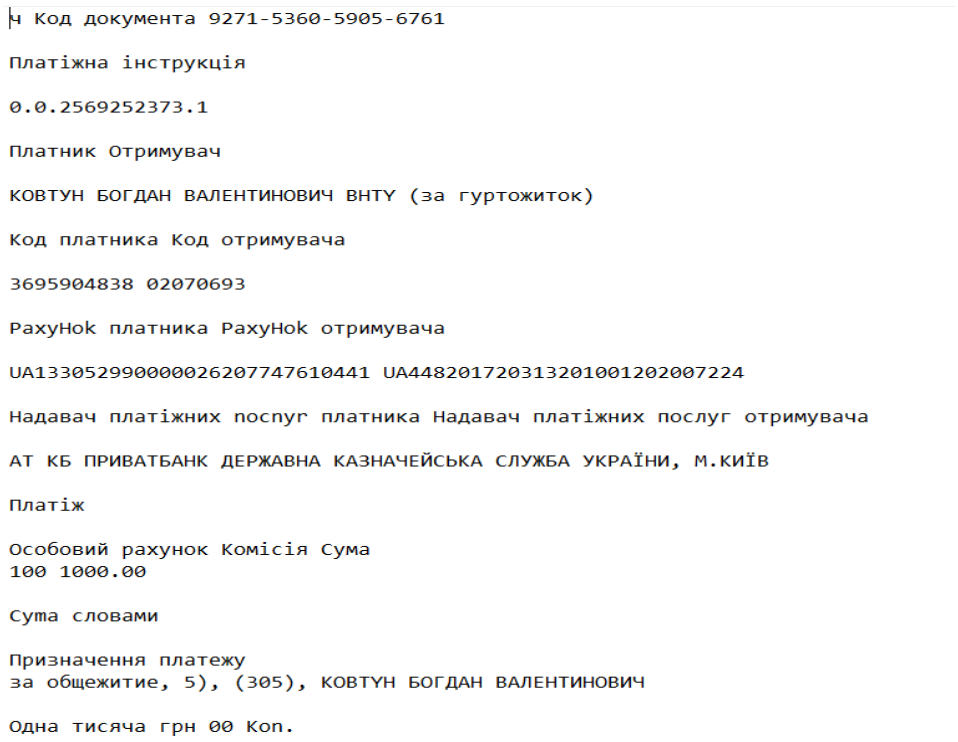


Рисунок 4.9 – Розпізнаний текст квитанції звичайним методом

З рисунку 4.9 можна зробити висновок, що рахунок отримувача та платника не є унікальним патерном, що ускладнює отримання необхідної інформації з розпізнаного тексту. Також для картинок проплат з різною роздільністю текст може міняти свою структуру, що може призвести до отримання неправильної інформації з розпізнаного тексту. Також сума проплати не має унікального патерну, оскільки у розпізнаному тексті присутні два числа. Тому скористатись Regex та отримати необхідну інформацію не є можливим, бо можливе розпізнання неправильної інформації, що є гіршим ніж не розпізнання інформації, бо неправильно розпізнана інформація може ввести в оману і привести до поганих наслідків для гуртожитку. Кращим рішенням є розроблюваний метод, що розпізнає текст лише у тих областях, де потрібна інформація має унікальний патерн, що унеможливлює розпізнання неправильної інформації. На рисунку 4.10 зображено три області розпізнання тексту для: кінцевого рахунку, номеру квитанції, суми проплати, розпізнання виконано розроблюваним методом.

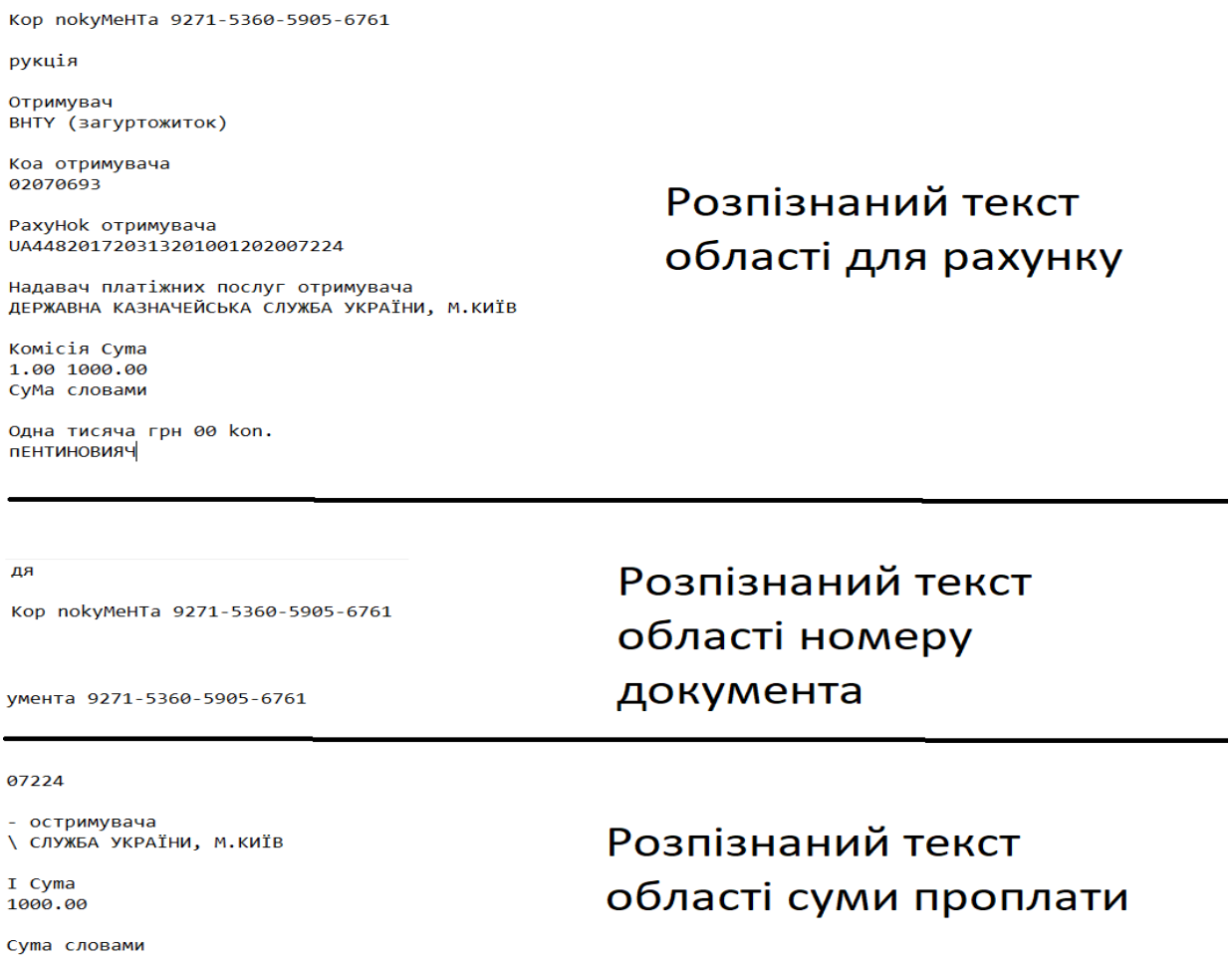


Рисунок 4.10 – Розпізнаний текст квитанції розроблюваним методом

Можна зробити висновок, що розроблюваний метод краще за звичайний, оскільки розроблюваний метод розпізнає текст по області де потрібна інформація буде мати унікальний паттерн. Розпізнаний текст розроблюваним методом має лише одне число у тексті, а розпізнаний текст звичайним методом має два числа. Можна зробити висновок, що потрібна інформація має унікальний паттерн при розпізнанні розроблюваним методом і тому цю інформацію легко отримати з тексту за допомогою Regex. Також розпізнання тексту зображення є не завжди надійним, оскільки розпізнаний текст буде мати різну структуру. Текст розпізнаних однакових зображень з різною роздільною здатністю може мати різну структуру і тому потрібна інформація повинна мати унікальний паттерн.

Також розроблюваний метод дозволяє розпізнавати проплати різних банків. Для кожного типу проплати потрібно створити власну стратегію і задати

параметри областів та Regex вирази для розпізнання потрібної інформації.

Також для точного порівняння розроблюваного методу зі звичайним методом розпізнання суми проплати студента було підготовлено 10 зображень проплат банку «ПриватБанк», зображення проплат включають електронний та паперовий варіант проплати. Результати порівняння методів розпізнання зображені на рисунку 4.11.

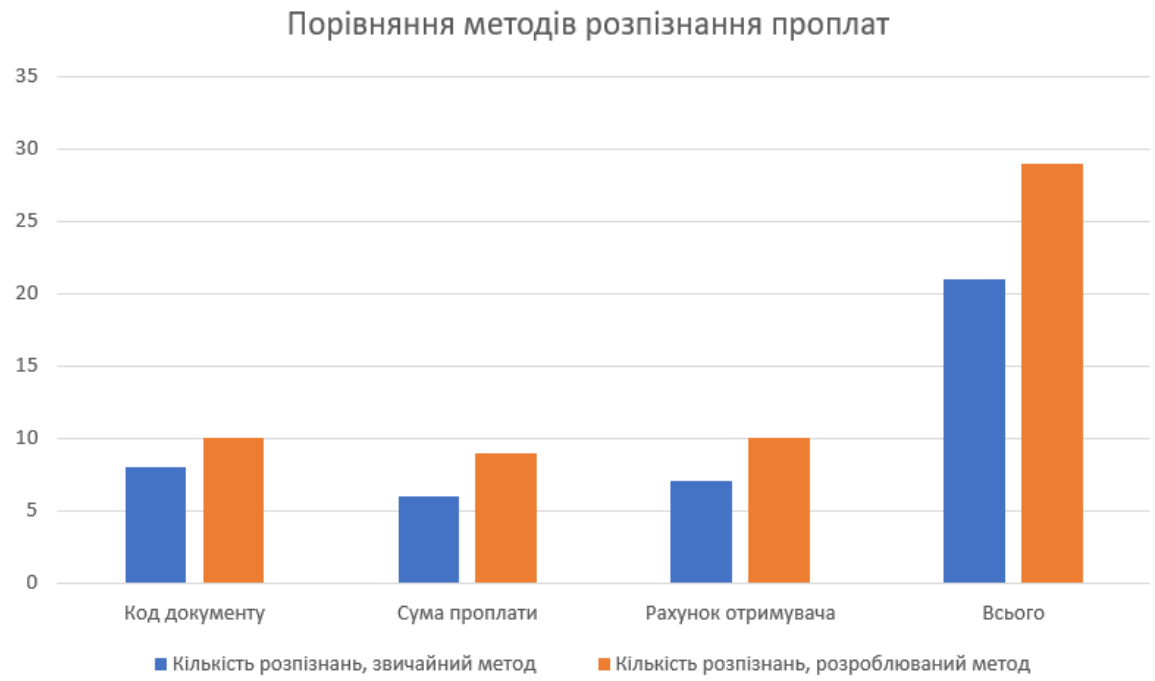


Рисунок 4.11 – Порівняння методів розпізнання проплат

Як показав рисунок розроблюваний метод розпізнав 29 пунктів необхідної інформації з 30, а у той час звичайний метод розпізнання розпізнав 21 пункт. Звичайний метод розпізнав 70% необхідної інформації, а розроблюваний метод розпізнав 96%. Розроблюваний метод розпізнання проплати розпізнав інформацію на 26% точніше.

Отже розроблюваний метод розпізнання суми проплати квитанції кращий за аналог, оскільки дозволяє розпізнавати текст в певній області, що дає можливість полегшити отримання необхідної інформації з картинки та реалізувати можливість розпізнання для різних банків. Також розроблюваний метод розпізнає інформацію з проплат точніше за звичайний метод.

4.4 Розробка інструкції користувача

Програмне забезпечення є веб-застосунком, тобто доступ до застосунку можливий за посиланням https://diplomnafrontend.azurewebsites.net/. Після переходу за посиланням користувач потрапить на головну сторінку сайту. Для отримання доступу до сайту потрібно мати інтернет та веб-браузер. Також для отримання доступу до функціоналу сайту потрібно мати створеного користувача, що буде мати певну роль. Студент та керуючий персонал будуть мати різний інтерфейс, оскільки мають різний функціонал. Головна сторінка сайту зображена на рисунку 4.12.

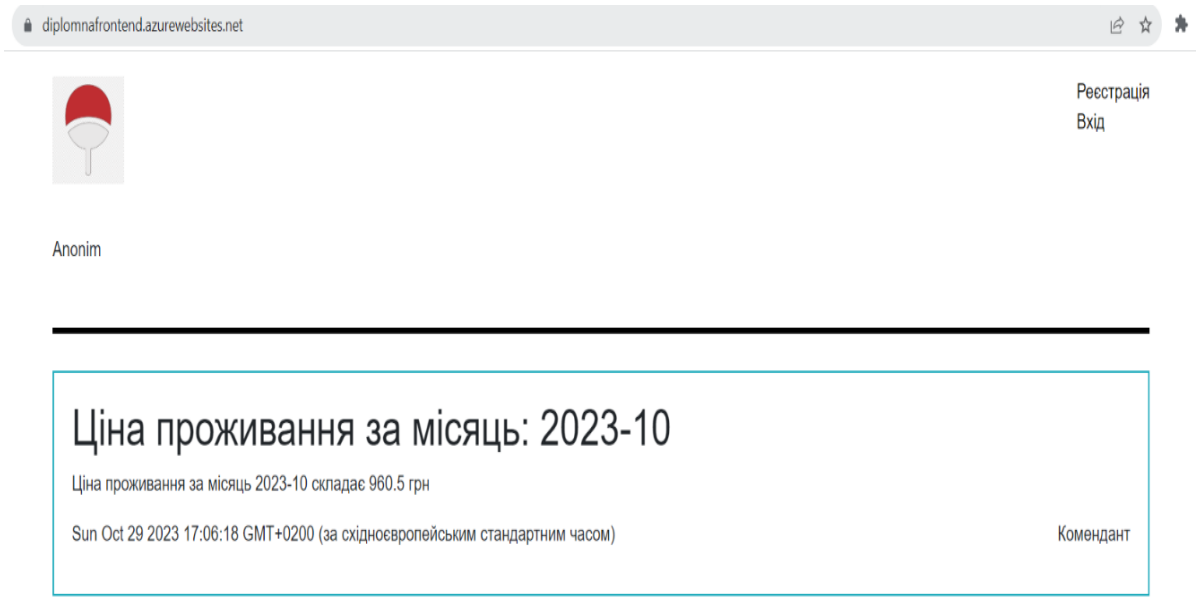


Рисунок 4.12 – Головна сторінка сайту

На головній сторінці сайту користувач бачить новини гуртожитку, для того щоб продовжити роботу із застосунком користувачу потрібно здійснити вхід. Також на головній сторінці сайту присутнє повідомлення «Anonim», що означає що користувач ще не здійснив вхід до сайту і йому потрібно авторизуватись. Також на головній сторінці присутня кнопка Реєстрація, але вона не працює, оскільки, щоб отримати доступ до застосунку потрібно вже мати зареєстрованого користувача. Реєструвати користувачів може лише керуючий персонал, бо потрібно зробити неможливим реєстрацію користувачів, які не

можуть приживати у гуртожитку університету. Для цього потрібно натиснути кнопку «Вхід» у правому верхньому кутку. Після натиснення користувача буде перекинуто на сайт аутентифікації та авторизації і користувачу буде потрібно ввести логін та пароль. Сторінка сайту авторизації зображена на рисунку 4.13.

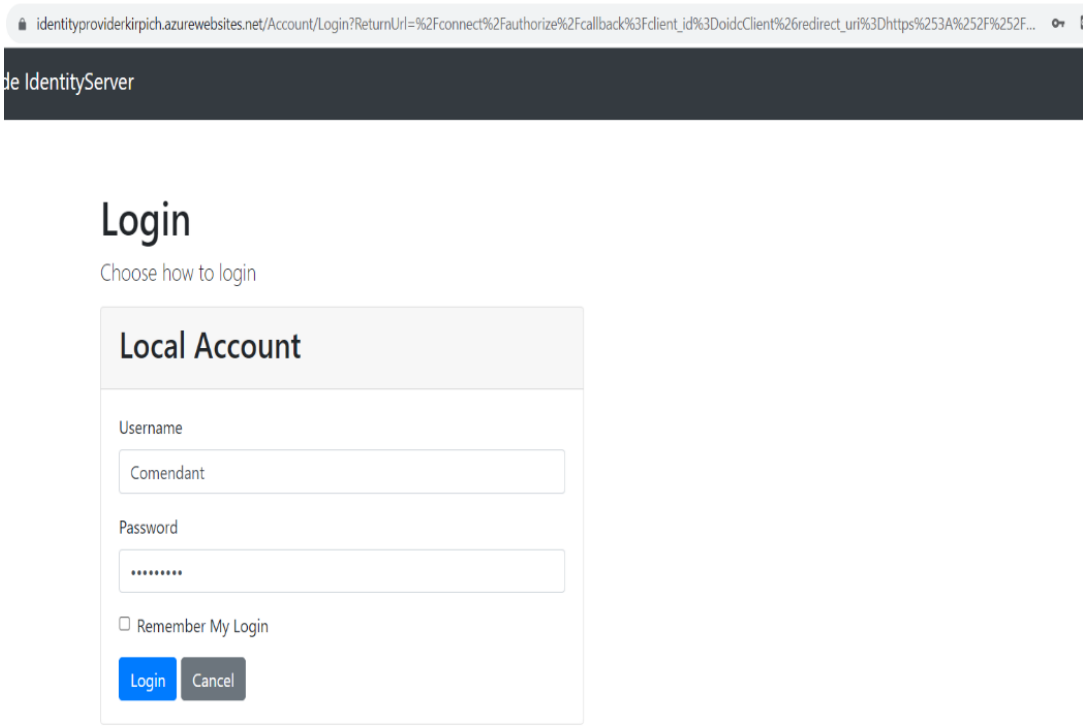


Рисунок 4.13 – Сторінка сайту авторизації

Після успішного введення логіну та паролю і натисненню кнопки «Login» користувача буде перекинуто на сайт гуртожитку. Після переходу на сайт гуртожитку користувач буде бачити останні новини з гуртожитку. Керуючий персонал також буде мати можливість додати нову новину у список новин. Після додавання нової новини кожен користувач зможе побачити нову новину при заході на головну сторінку сайту. На головній сторінці гуртожитку для керуючого персоналу присутні кнопки керування: «Студенти», що відповідає за перехід до сторінки списку студентів, «Реєстрація», що відповідає за перехід до сторінки реєстрації нових користувачів, «Кімнати», що відповідає за перехід до сторінки кімнат, «Профіль», що відповідає за перехід у сторінку профілю для авторизованого користувача та «Ціна проживання за місяць», що відповідає за

перехід на сторінку встановлення ціни за місяць проживання у гуртожитку. Головна сторінка гуртожитку для керуючого персоналу зображена на рисунку 4.14.

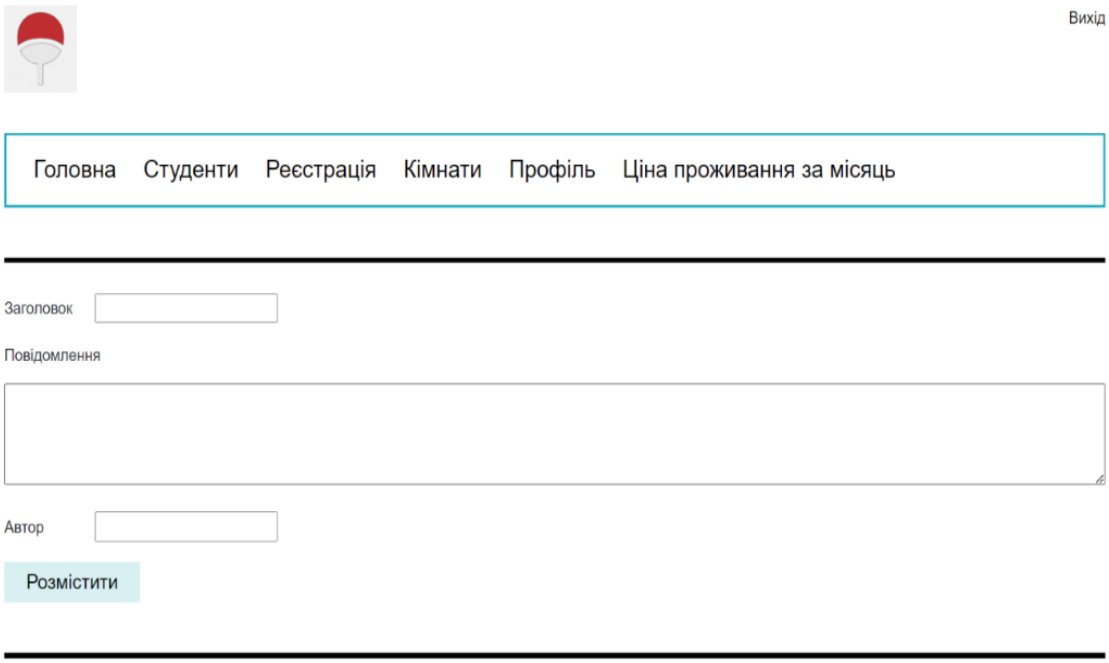


Рисунок 4.14 – Головна сторінка сайту гуртожитку для керуючого персоналу

На головній сторінці для студента також будуть присутні новини як і для всіх інших користувачів. Студент може з головного вікна перейти на сторінки: «Проплата», що відповідає за сторінку проплат дестудент може завантажити проплату та розрахувати поточний борг для себе, «Профіль», що відповідає за перехід на сторінку профілю де користувач може заповнити всю необхідну інформацію про себе, «Заява на проживання», що відповідає за можливість переходу на сторінку де студент може створити нову заяву на проживання та згенерувати заяву на проживання. «Комендант», що відповідає за перехід на сторінку коменданта, але на даний момент перехід недоступний, бо сторінка ще не реалізована. На рисунку 4.15 зображено рисунок головної сторінки сайту гуртожитку для студента.

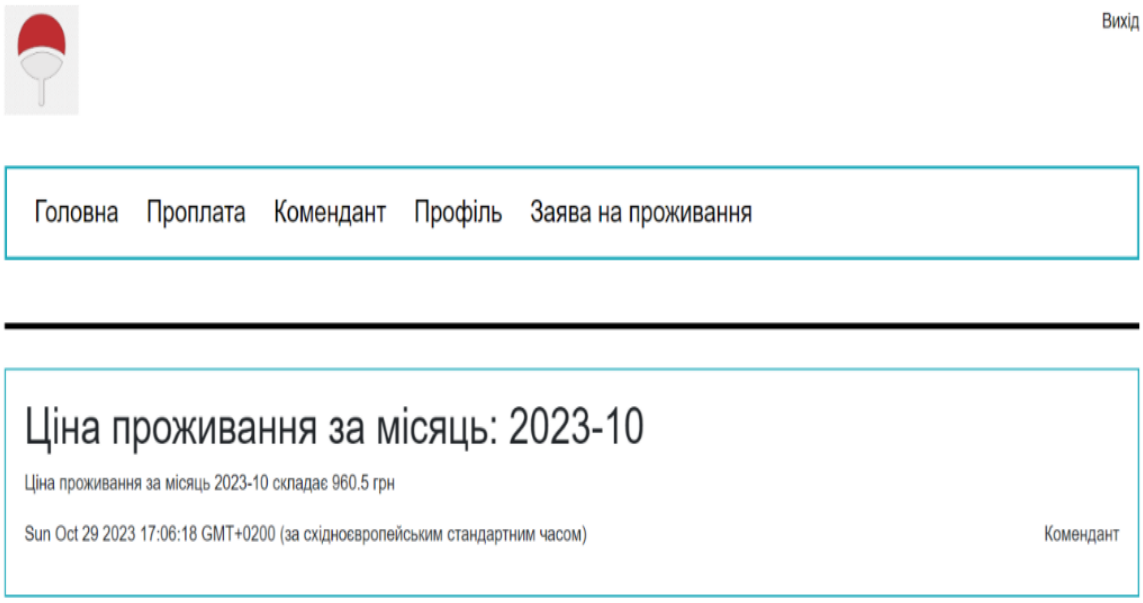


Рисунок 4.15 – Головна сторінка сайту гуртожитку для студента

Після входу у систему користувач може користуватись застосунком. Але також користувач може вийти з системи скориставшись кнопкою «Вихід». Після натиснення кнопки «Вихід» користувач буде переадерсований на сайт аутентифікації, де йому потрібно буде підтвердити чи виходити з системи. Після підтвердження користувача буде переадресовано на головну сторінку сайту для гуртожитку. На рисунку 4.16 зображено сторінку підтвердження виходу.

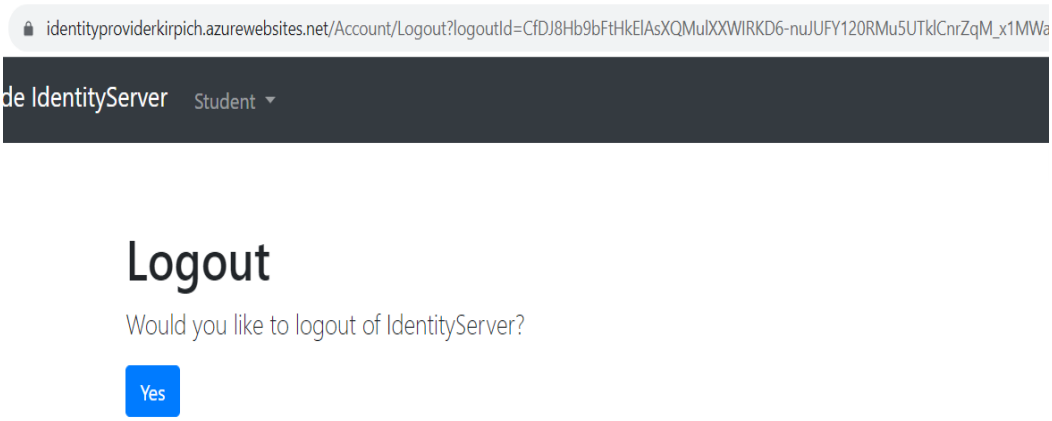


Рисунок 4.16 – Сторінка підтвердження виходу з системи

Оскільки для входу у застосунок, користувач вже повинен мати зареєстрованого юзера, то керуючому персоналу потрібно надати можливість створювати нових юзерів. Для створення нового юзера потрібно вказати роль користувача. Доступні ролі для застосунку: comendant, student, dean, passportHolder. Також створювати нових користувачів може лише керуючий персонал. Також потрібно вказати username для нового користувача, цей username повинен містити лише латинські літери та цифри. Також потрібно заповнити поле для паролю пароль можна згенерувати на іншому сайті генераторів паролю, скопіювати та вставити пароль у поле для паролю, також можна придумати та написати пароль у полі для паролю. На рисунку 4.17. зображено сторінку створення нового користувача.

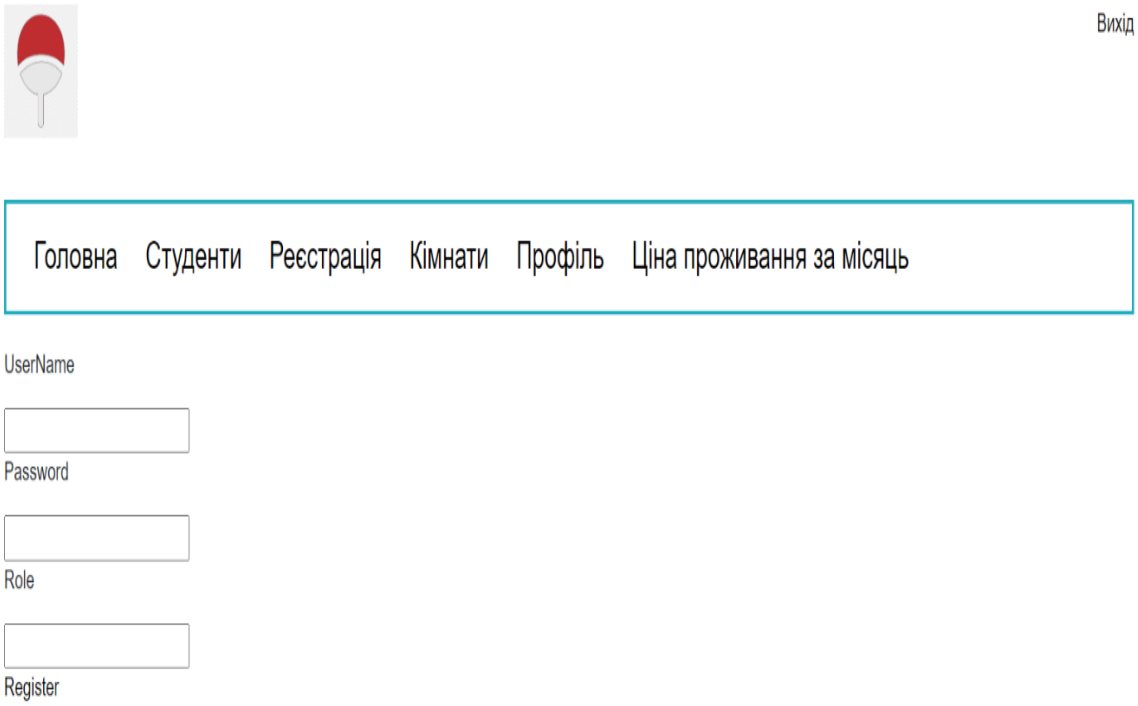


Рисунок 4.17 – Сторінка створення користувача

Отже, була створена інструкція користувача, в якій описано послідовність дій, що потрібно зробити щоб почати використовувати застосунок для підвищення ефективності процесу управління гуртожитку університету.

4.5 Вимоги до персонального комп’ютера

Розроблений програмний продукт є веб-застосунком тому може бути доступним для будь-якого пристрою, що підтримує веб-браузер. Застосунок може використовуватись як десктопними так і мобільними девайсами.

Отже, у таблиці 4.1 зображено мінімальні характеристики комп’ютера, а у таблиці 4.2 зображено рекомендовані характеристики комп’ютера.

Таблиця 4.1 – Мінімальні характеристики

Тип процесора	32-розрядний (x86) 1,4 ГГц
Об’єм оперативної пам’яті	2 ГБ
Місце на жорсткому диску	2 ГБ
Графічний пристрій	Інтегрований графічний пристрій, сумісний з DirectX9
Операційна система	Windows 10

Таблиця 4.2 – Рекомендовані характеристики

Тип процесора	64-розрядний (x64) 2,4 ГГц
Об’єм оперативної пам’яті	4 ГБ
Місце на жорсткому диску	2 ГБ
Графічний пристрій	Інтегрований графічний пристрій, сумісний з DirectX11
Операційна система	Windows 10

4.6 Висновки

Отже, було проведено тестування функціоналу програмного забезпечення для підвищення ефективності процесу управління гуртожитком університету з використанням методу «чорна скринька». Також було розроблено тест-кейси і по цим тест-кейсам була перевірена повна працездатність застосунку.

Було проведено тестування розроблюваного методу розпізнання суми проплати з чеків, що завантажуються студентом.

Було розроблено інструкцію користувача та визначено мінімальні та рекомендовані характеристики комп’ютера.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота за темою «Розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

Метою проведення комерційного і технологічного аудиту дослідження за темою «Розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в

результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1 [29].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою

Продовження таблиці 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

7	Активна конкуренція великих компаній на	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці. Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки

експертами зображено на таблиці 5.2.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	5	4	4
2. Ринкові переваги (наявність аналогів)	4	3	3
3. Ринкові переваги (ціна продукту)	4	4	4
4. Ринкові переваги (технічні властивості)	3	4	3
5. Ринкові переваги (експлуатаційні витрати)	4	3	3
6. Ринкові перспективи (розмір ринку)	3	4	3
7. Ринкові перспективи (конкуренція)	3	3	4
8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	3	3	4
10. Практична здійсненність (необхідність нових матеріалів)	3	3	3
11. Практична здійсненність (термін реалізації)	3	3	4
12. Практична здійсненність (розробка документів)	3	3	3
Сума балів	41	40	41
Середньоарифметична сума балів $СБ_c$	40,7		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 [29].

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка методів і програмних засобів для підвищення ефективності

процесу управління гуртожитком університету» становить 40,7 балів, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

5.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов'язані з проведенням науково-дослідної роботи на тему «Розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [29]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \qquad (5.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дні;

T_p – середнє число робочих днів в місяці, T_p=22 дні.

$$З_о = 20000{,}00 \cdot 50 / 22 = 45454{,}55 \text{грн.}$$

Проведені розрахунки зведемо до таблиці 5.4.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	20000	909,09	50	45454,55
Інженер-розробник програмного забезпечення	15000	681,82	50	34090,91
Консультант	15000	681,82	5	3409,09
Всього				82954,55

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i \,, \tag{5.2}$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} \,, \tag{5.3}$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6700,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду [29];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$T_р$ – середнє число робочих днів в місяці, приблизно $T_р$ = 22 дні;

$t_{зм}$ – тривалість зміни, год.

$С_1 = 6700,00 \cdot 1,10 \cdot 1,65 / (22 \cdot 8) = 69,09$ грн.
$З_{р1} = 69,09 \cdot 4,00 = 276,38$ грн.

Величина витрат на основну заробітну плату робітників зображена у таблиці 5.5.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Підготовка робочого місця дослідника	4	2	1,1	69,09	276,38
Інсталяція програмного забезпечення	3	5	1,7	106,78	320,34
Тестування системи	4	2	1,1	69,09	276,38
Всього					873,09

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод} = (82954,55 + 873,09) \cdot 11 / 100\% = 9221,04$грн.

5.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (82954,55+ 873,09+ 9221,04) · 22 / 100% = 20470,71 грн.

5.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}, \quad (5.6)$$

де $Н_j$ – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 ... 1,15*);

B_j – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$М_1 = 2 \cdot 210{,}00 \cdot 1{,}1 - 0{,}000 \cdot 0{,}00 = 462{,}0$ грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір А4-500-80	210	2	0	0	462

Продовження таблиці 5.6 – Витрати на матеріали

Канцелярське приладдя (набір)	175	2	0	0	385
Всього					847

5.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету» відсутні.

5.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Витрати на спецустаткування, які використовують при проведенні НДР на тему «Розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету» відсутні.

5.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (4.7)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (K_i = *1,10…1,12);*

k – кількість найменувань програмних засобів.

$B_{прг} = 3000 \cdot 1 \cdot 1{,}1 = 3360$ *грн.*

Отримані результати зведемо до таблиці 5.7:

Таблиця 5.7– Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Microsoft Azure	1	3000	3360
Всього			3360

5.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.8)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (60000{,}00 \cdot 3) / (5 \cdot 12) = 3000$ *грн.*

Проведені розрахунки зведемо до таблиці 5.8.

Таблиця 5.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний компютер - 3шт	60 000	5	3	3000,00
Microsoft Azure	3 360	3	3	280,00
Робоче місце дослідника	15000	5	3	750,00
Оргтехніка	6000	4	2	250,00
Приміщення лабораторії	212000	22	3	2409,09
ОС Windows 11, 3шт	28500	3	3	2375,00
Прикладний пакет Microsoft Office 2019,3	15000	3	3	1250,00
Всього				10314,09

5.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (5.9)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e$ = 7,50 грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_е = 0,2 \cdot 480,0 \cdot 7,50 \cdot 0,95 / 0,97 = 2820,62$ *грн.*

Проведені розрахунки зведемо до таблиці 5.9.

Таблиця 5.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний компютер (2 шт)	0,2	400	1175,26
Персональний компютер (1 шт)	0,2	280	411,34
Робоче місце дослідника	0,15	400	440,72
Оргтехніка	0,45	20	66,11
Всього			2093,43

5.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \qquad (5.10)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$B_{св} = (82954,55 + 873,09) \cdot 20 / 100\% = 16765,53$*грн.*

5.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \qquad (5.11)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 40%.

$В_{сп}$ = (82954,55+ 873,09) · 40 / 100% = 33531,06грн.

5.2.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \qquad (5.12)$$

де *$Н_{ів}$* – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 60%.

$І_в$ = (82954,55+ 873,09) · 60 / 100% = 50296,58грн.

5.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \qquad (5.13)$$

де *$Н_{нзв}$* – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 120%.

$В_{нзв}$ = (82954,55+ 873,09) · 100 / 120% = 100 593,17 грн.

Витрати на проведення науково-дослідної роботи розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.14)$$

$B_{заг=}$ *331320,24грн.*

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.15)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9.

ЗВ = 331320,24/ 0,9 = 368133,60грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

1-й рік – 6 користувачів;

2-й рік –7 користувачів;

3-й рік – 4 користувачів.

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 5

користувачів;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 200000,00 грн;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 80000,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [29]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.16)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2023 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо ρ =40%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (5 \cdot 80000,00 + 280000 \cdot 6) \cdot 0,83 \cdot 0,4 \cdot (1\text{-}0,18/100\%) =$ 568305,92 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (5 \cdot 80000,00 + 280000 \cdot (6+7)) \cdot 0,83 \cdot 0,4 (1\text{-}0,18/100\%) =$ 1103824,96 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (5 \cdot 80000,00 + 280000 \cdot (6+7+4)) \cdot 0,83 \cdot 0,4 \cdot (1\text{-}0,18/100\%) =$ 1409835,84грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.17)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ=0,15;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП$ = 568305,92/(1+0,15)^1+ 1103824,96/(1+0,15)^2+ 1409835,84/(1+0,15)^3= 2255819,26грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (5.18)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 368133,60грн.

$PV = k_{інв} \cdot ЗВ$ = 2· 368133,60= 736267,2041грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \quad (5.19)$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 2255819,26грн;

PV – теперішня вартість початкових інвестицій, 736267,2041грн.

$E_{абс} = ПП - PV$ = 2255819,26 - 736267,2041= 1519552,06грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.20)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 1519552,06 грн;

PV – теперішня вартість початкових інвестицій, 736267,2041грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1$ = (1+1519552,06/736267,2041)$^{1/3}$= 0,45.

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (5.21)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,18.

$\tau_{мін}$ = 0,1+0,18 = 0,28 < 0,45 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у

впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{5.22}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 0,45 = 2,2 року.

$T_{ок} <$ 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.4 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету» становить 40,7 балів, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

Також термін окупності становить 2,2 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету».

ВИСНОВКИ

У магістерській кваліфікаційній роботі було розроблено методи і програмні засоби для підвищення ефективності процесу управління гуртожитку університету.

Було проведено аналіз предметної області та проаналізовано існуючі аналоги, було розглянуто переваги та недоліки аналогів та проведено їх порівняння з розроблюваним застосунком. За результатами порівняння було вирішено, що розробка власного рішення є доцільною. Було виконано постановку задач для розробки.

Удосконалено метод розпізнавання суми проплати, що завантажується студентом, у якому, на відміну від існуючих рішень, розпізнавання тексту відбувається з використанням декількох стратегій і за допомогою Regex, що дало можливість підвищити точність розпізнавання суми оплати з чеків різних банків на 26%.

Подальшого розвитку отримав метод автоматизації обліку боргу студента, що на відміну від відомих рішень, динамічно визначає борг шляхом встановлення комендантом помісячної вартості проживання, обліку місяців, що прожив студент у гуртожитку, та суми фактичних проплат студента, що дало можливість прискорити визначення боргу динамічно з появою нових даних.

Подальшого розвитку отримав метод генерації документів, який на відміну від інших рішень, генерує документи з релевантною інформацією, яка отримана з профілів користувачів, що дало можливість повністю автоматизувати процес створення документів.

Проведено варіантний аналіз і обґрунтовано вибір засобів реалізації застосунку. Було обрано основну мову програмування для розробки застосунку TypeScript. Розробка застосунку відбувалася за допомогою середовища розробки Visual Studio Code. База даних, що використовувалась MongoDB. Також було використано бібліотеку React для розробки інтерфейсу застосунку. Було розроблено застосунок для підвищення ефективності процесу управління

гуртожитку університету та зроблено аналіз коду застосунку.

Було проведено тестування розробленого застосунку з використанням методу «чорної скриньки». За результатами тестування було визначено, що застосунок відповідає критеріям якості та була доведена його працездатність. Також було проведено дослідження методу розпізнання суми проплати, в результаті чого було доведено працездатність та ефективність методу. Було розроблено інструкцію користувача. Також було представлено мінімальні та рекомендовані характеристики комп’ютера для використання розробленого застосунку.

Отримані в магістерській кваліфікаційній роботі наукові та практичні положення застосовано в системі для підвищенні ефективності процесу управління гуртожитку університету.

Результати роботи доповідались на науково-технічних конференціях та опубліковані у трьох наукових публікаціях.

Було проведено комерційний аналіз розробки, який довів, що розроблений застосунок за своїми характеристиками є кращим за аналогічні програмні застосунки та є перспективною розробкою. Застосунок має кращі функціональні показники, що робить його конкурентоспроможним на ринку.

Задачі магістерської кваліфікаційної роботи було виконано у повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Patton L. D. Student Development in College: Theory, Research, and Practice [Електронний ресурс] / Lori D. Patton, Kristen A. Renn. – London : Jossey-Bass, 2016. – 560 с. – Режим доступу: https://www.amazon.com/Student-Development-College-Research-Practice/dp/1118821815 (дата звернення: 29.11.2023). – Назва з екрана.

2. Rutherford D. G. Hotel Management and Operations [Електронний ресурс] / Denney G. Rutherford. – Hoboken : Wiley, 2010. – 512 с. – Режим доступу: https://www.amazon.com/Hotel-Management-Operations-Michael-OFallon/dp/0470177144 (дата звернення: 29.11.2023). – Назва з екрана.

3. Collins G. R. Hospitality Information Technology: Learning How to Use It. [Електронний ресурс] / Galen R. Collins. – Dubuque : Kendall Hunt Publishing, 2017. – 214 с. – Режим доступу: https://www.amazon.com/Hospitality-Information-Technology-Learning-How/dp/1524917850 (дата звернення: 29.11.2023). – Назва з екрана.

4. Ковтун Б.В. Розпізнавання чеку проплати за допомогою технології OCR. Матеріали XVI міжнародної науково-практичної конференції «Інформаційні технології і автоматизація – 2023» [Електронний ресурс] / Б.В. Ковтун, О.В. Романюк. – Режим доступу: https://ontu.edu.ua/download/konfi/2023/Collection-of-abstracts-of-the-conference-ITIA-2023.pdf (дата звернення: 29.11.2023). – Назва з екрана.

5. Ковтун Б. Розробка методу розпізнання суми проплати з чеків різних банків. Матеріали міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» [Електронний ресурс] / Б.В. Ковтун, О.В. Романюк. – Режим доступу: https://press.vntu.edu.ua/index.php/vntu/catalog/book/56 (дата звернення: 29.11.2023). – Назва з екрана.

6. Ковтун Б. Аналіз методів розпізнавання тексту на зображеннях. Матеріали LII науково-технічної конференції підрозділів Вінницького

національного технічного університету (НТКПВНТУ–2023) [Електронний ресурс] / Б.В. Ковтун, О.В. Романюк. – Режим доступу: https://press.vntu.edu.ua/index.php/vntu/catalog/view/788/1373/2632-1 (дата звернення: 29.11.2023). – Назва з екрана.

7. Solomon M. The Heart of Hospitality: Great Hotel and Restaurant Leaders Share Their Secrets Hardcover [Електронний ресурс] / Micah Solomon. – New York : SelectBooks, 2016. – 192 с. – Режим доступу: https://www.amazon.com/Heart-Hospitality-Restaurant-Leaders-Secrets/dp/1590793781 (дата звернення: 29.11.2023). – Назва з екрана.

8. Hotel Automation: Trends, Tools, and Tips to Know [Електронний ресурс]. – Режим доступу: https://www.cvent.com/en/blog/hospitality/hotel-automation (дата звернення: 29.11.2023). – Назва з екрана.

9. Blokdyk G. Optical character recognition A Complete Guide Kindle [Електронний ресурс] / Gerardus Blokdyk. – London : 5STARCooks, 2018. – 162 с. – Режим доступу: https://www.amazon.com/Optical-character-recognition-Complete-Guide-ebook/dp/B07DV6MFCS (дата звернення: 29.11.2023). – Назва з екрана.

10. Obaidullah S. M. Document Processing Using Machine Learning [Електронний ресурс] / Sk Md Obaidullah. – London : Chapman and Hall/CRC, 2019. – 168 с. – Режим доступу: https://www.amazon.com/Document-Processing-Using-Machine-Learning/dp/036721847X (дата звернення: 29.11.2023). – Назва з екрана.

11. Wellman D. Ultimate Typescript Handbook: Build, scale and maintain Modern Web Applications with Typescript [Електронний ресурс] / Dan Wellman. – Delhi : Orange Education Pvt Ltd, 2023. – 436 с. – Режим доступу: https://www.amazon.com/Ultimate-Typescript-Handbook-maintain-Applications/dp/9388590783 (дата звернення: 29.11.2023). – Назва з екрана.

12. Haverbeke M. Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming [Електронний ресурс] / Marijn Haverbeke. – San Francisco : No Starch Press, 2018. – 472 с. – Режим доступу: https://www.amazon.com/Eloquent-

JavaScript-3rd-Introduction-Programming/dp/1593279507 (дата звернення: 29.11.2023). – Назва з екрана.

13. Bloch J. Effective Java [Електронний ресурс] / Joshua Bloch. – Boston : Addison-Wesley Professional, 2017. – 416 с. – Режим доступу: https://www.amazon.com/Effective-Java-Joshua-Bloch/dp/0134685997 (дата звернення: 29.11.2023). – Назва з екрана.

14. Stroustrup B. Tour of C++, A (C++ In-Depth Series) [Електронний ресурс] / Bjarne Stroustrup. – Boston : Addison-Wesley Professional, 2022. – 320 с. – Режим доступу: https://www.amazon.com/Tour-C-Bjarne-Stroustrup/dp/0136816487 (дата звернення: 29.11.2023). – Назва з екрана.

15. Strauss D. Getting Started with Visual Studio 2022: Learning and Implementing New Features [Електронний ресурс] / Dirk Strauss. – New-York : Apress, 2022. – 331 с. – Режим доступу: https://www.amazon.com/Getting-Started-Visual-Studio-2022/dp/1484289218 (дата звернення: 29.11.2023). – Назва з екрана.

16. Rosca S. WebStorm Essentials: Build efficient HTML, CSS and JavaScript applications using the powerful WebStorm IDE [Електронний ресурс] / Stefan Rosca. – Birmingham : Packt Publishing, 2015. – 184 с. – Режим доступу: https://www.amazon.com/WebStorm-Essentials-Stefan-Rosca/dp/1785286951 (дата звернення: 29.11.2023). – Назва з екрана.

17. Johnson B. Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers [Електронний ресурс] / Bruce Johnson. – Hoboken : Wiley, 2019. – 192 с. – Режим доступу: https://www.amazon.com/Visual-Studio-Code-End-End/dp/1119588189 (дата звернення: 29.11.2023). – Назва з екрана.

18. Phaltankar A. MongoDB Fundamentals: A hands-on guide to using MongoDB and Atlas in the real world [Електронний ресурс] / Amit Phaltankar. – Birmingham : Packt Publishing, 2020. – 748 с. – Режим доступу: https://www.amazon.com/MongoDB-Workshop-Interactive-Approach-Learning/dp/1839210648 (дата звернення: 29.11.2023). – Назва з екрана.

19. CouchDB Team. CouchDB 2.0 Reference Manual [Електронний ресурс] / CouchDB Team. – London : Samurai Media Limited, 2015. – 416 с. – Режим доступу:

https://www.amazon.com/CouchDB-2-0-Reference-Manual-Team/dp/9888381652 (дата звернення: 29.11.2023). – Назва з екрана.

20. Shaikh S. Mastering RethinkDB [Електронний ресурс] / Shahid Shaikh. – Birmingham : Packt Publishing, 2016. – 463 с. – Режим доступу: https://www.amazon.com/Mastering-RethinkDB-Shahid-Shaikh/dp/1786461072 (дата звернення: 29.11.2023). – Назва з екрана.

21. Tidwell J. Designing Interfaces: Patterns for Effective Interaction Design [Електронний ресурс] / Jenifer Tidwell. – Sebastopol : O'Reilly Media, 2020. – 599 с. – Режим доступу: https://www.amazon.com/Designing-Interfaces-Patterns-Effective-Interaction/dp/1492051969 (дата звернення: 29.11.2023). – Назва з екрана.

22. Banks A. Learning React: Modern Patterns for Developing React Apps [Електронний ресурс] / Alex Banks. – Sebastopol : O'Reilly Media, 2020. – 307 с. – Режим доступу: https://www.amazon.com/Learning-React-Modern-Patterns-Developing/dp/1492051721 (дата звернення: 29.11.2023). – Назва з екрана.

23. Seshadri S. Angular: Up and Running: Learning Angular, Step by Step [Електронний ресурс] / Shyam Seshadri. – Sebastopol : O'Reilly Media, 2018. – 309 с. – Режим доступу: https://www.amazon.com/Angular-Up-Running-Learning-Step/dp/1491999837 (дата звернення: 29.11.2023). – Назва з екрана.

24. Macrae C. Vue.js: Up and Running: Building Accessible and Performant Web Apps [Електронний ресурс] / Callum Macrae. – Sebastopol : O'Reilly Media, 2018. – 171 с. – Режим доступу: https://www.amazon.com/Vue-js-Running-Building-Accessible-Performant/dp/1491997249 (дата звернення: 29.11.2023). – Назва з екрана.

25. White-Box Testing [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/software-engineering-white-box-testing/ (дата звернення: 29.11.2023). – Назва з екрана.

26. White/black/grey box-тестування. [Електронний ресурс]. – Режим доступу: https://qalight.ua/baza-znaniy/white-black-grey-box-testuvannya (дата звернення: 29.11.2023). – Назва з екрана.

27. Black-Box Testing [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/software-engineering-black-box-testing/ (дата звернення: 29.11.2023). – Назва з екрана.

28. About test-cases [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/test-case/ (дата звернення: 29.11.2023). – Назва з екрана.

29. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. Вінниця : ВНТУ, 2021. 42 с.

30. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с.

ДОДАТКИ

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач каф. ПЗ

д.т.н., проф. Романюк О. Н.

"19" вересня 2023 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету» за спеціальністю

121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

к.т.н., доц. Романюк О.В.

"19" вересня 2023 р.

Виконав:

студент гр.ЗПІ-22м Ковтун Б.В.

"19" вересня 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету».

Галузь застосування – автоматизація процесу управління гуртожитком університету.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ №247 від 18 вересня 2023 року ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є підвищення ефективності процесу управління гуртожитком університету шляхом розробки методів та програмних засобів для автоматизації процесів управління боргами, оплатами та інформацією про студента, а також генерації документів.

Призначення роботи – розробка методів і засобів для підвищення ефективності процесу управління гуртожитком університету.

4. **Вихідні дані для проведення НДР**

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

1. Patton L. D. Student Development in College: Theory, Research, and Practice [Електронний ресурс] / Lori D. Patton, Kristen A. Renn. – London : Jossey-Bass, 2016. – 560 с. – Режим доступу: https://www.amazon.com/Student-Development-College-Research-Practice/dp/1118821815 (дата звернення: 29.11.2023). – Назва з екрана.

2. Obaidullah S. M. Document Processing Using Machine Learning [Електронний ресурс] / Sk Md Obaidullah. – London : Chapman and Hall/CRC,

2019. – 168 с. – Режим доступу: https://www.amazon.com/Document-Processing-Using-Machine-Learning/dp/036721847X (дата звернення: 29.11.2023). – Назва з екрана.

3. Banks A. Learning React: Modern Patterns for Developing React Apps [Електронний ресурс] / Alex Banks. – Sebastopol : O'Reilly Media, 2020. – 307 с. – Режим доступу: https://www.amazon.com/Learning-React-Modern-Patterns-Developing/dp/1492051721 (дата звернення: 29.11.2023). – Назва з екрана.

4. Phaltankar A. MongoDB Fundamentals: A hands-on guide to using MongoDB and Atlas in the real world [Електронний ресурс] / Amit Phaltankar. – Birmingham : Packt Publishing, 2020. – 748 с. – Режим доступу: https://www.amazon.com/MongoDB-Workshop-Interactive-Approach-Learning/dp/1839210648 (дата звернення: 29.11.2023). – Назва з екрана.

5. Технічні вимоги

Вихідні дані до роботи: модель розробки – водоспадна; метод передачі повідомлень між сервісами – HTTP; система управління базами даних – MongoDB; вхідні дані – база даних з інформацією про студента, з інформацією про керуючий персонал, з підписами користувачів, з проплатами за проживання у гуртожитку завантажені студентами, з цінами за проживання за місяць у гуртожитку, з історією студентів проживання у гуртожитку; вихідні дані – розпізнана сума проплати та розрахований борг студента у вигляді текстового звіту, згенерована заява на проживання у гуртожитку у вигляді word документа; середовище розробки – Visual Studio Code; мова програмування – TypeScript.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

– пояснювальна записка до МКР;

– технічне завдання;

– 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 магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану питання та постановка задач дослідження	20.09. 2023 – 19.10.2023
2	Розробка методів підвищення ефективності процесу управління гуртожитком університету	20.10.2023 – 24.10.2023
3	Розробка алгоритмів роботи програмного застосунку	25.10.2023 – 19.11.2023
4	Розробка програмного забезпечення	20.11.2023 – 24.11.2023
5	Тестування програмного застосунку та дослідження ефективності запропонованого методу	25.11.2023 – 26.11.2023
6	Економічна частина	27.11.2023 – 28.11.2023
7	Оформлення матеріалів до захисту МКР	29.11.2023 – 01.12.2023

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б
Протокол перевірки МКР на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **«Розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету».**

Тип роботи: магістерська кваліфікаційна робота
Підрозділ: кафедра програмного забезпечення, ФІТКІ, ЗПІ – 22м
Науковий керівник: к.т.н. доц. Романюк О.В.

Unicheck	
Оригінальність	**95,9%**
Схожість	4,1%

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.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck.

Автор роботи _____ Ковтун Б.В.

Керівник роботи _____ Романюк О.В.

ДОДАТОК В

Лістинг коду

auth.module.ts

```
import { Module } from '@nestjs/common';
import { PassportModule } from '@nestjs/passport';
import { BearerTokenAuthGuard } from './bearer-token-auth-guard'; // Create this
strategy

@Module({
    imports: [PassportModule, BearerTokenAuthGuard],
    providers: [
        BearerTokenAuthGuard
    ],
})
export class AuthModule { }
```

bearer-token-auth-guard.ts

```
import { ExecutionContext, Injectable, UnauthorizedException, ForbiddenException }
from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';
import jwt from 'jsonwebtoken';
import jwksClient from 'jwks-rsa';
import https from 'https'
import { Reflector } from '@nestjs/core'
import { CustomRequest } from './customRequest';

@Injectable()
export class BearerTokenAuthGuard extends AuthGuard('bearer') {

    constructor(private reflector: Reflector) {
        super()
    }

    async canActivate(context: ExecutionContext) {
        const request = context.switchToHttp().getRequest<CustomRequest>();
        const token = this.extractTokenFromRequest(request);

        if (!token) {
            console.log('Bearer token not found')
            throw new UnauthorizedException('Bearer token not found');
        }

        await this.validateToken(token, context);

        const userAndRole = this.validateIsHavePermissionAndGetUserAndRole(token,
```

```
context);

        request.userId = userAndRole.userId;
        request.role = userAndRole.role;

        return true;
    }

    private validateIsHavePermissionAndGetUserAndRole(token: string, context:
ExecutionContext) {

        const decodedToken = jwt.decode(token, { complete: true });

        const tokenRole = decodedToken['payload']['role'];

        const roles = this.reflector.get<string[]>('roles', context.getHandler());

        const scopes = this.reflector.get<string[]>('scopes',
context.getHandler());


        if (this.authorizeToken(decodedToken, tokenRole, roles, scopes) == false) {
            console.log('forbidden')
            throw new ForbiddenException('Forbidden');
        }



        const tokenUserId = decodedToken['payload']['sub'];

        return {
            role: tokenRole,
            userId: tokenUserId
        }
    }

    private authorizeToken(decodedToken, tokenRole, roles: string[], scopes:
string[]) {
        if (decodedToken['payload']['client_id'] === 'oidcClient') {
            return roles.includes(tokenRole);
        }
        if (scopes) {
            let authorized = false;
            decodedToken['payload']['scope'].forEach(scope => {
                if (scopes.includes(scope)) {
                    authorized = true;
                }
                return;
            });
            return authorized;
        }
```

```
        return false;
    }

    private async validateToken(token: string, context: ExecutionContext):
Promise<void> {
        try {
            const decodedToken = jwt.decode(token, { complete: true });
            const kid = decodedToken['header']['kid'];
            console.log(process.env.JWT_URL)
            const key = await this.getKey(process.env.JWT_URL, kid);
            jwt.verify(token, key);
        } catch (exc) {
            console.log('Invalid bearer token')
            throw new UnauthorizedException('Invalid bearer token');
        }
    }

    private async getKey(jwksUri, kid): Promise<string> {
        let requestAgent = new https.Agent({
            rejectUnauthorized: false
        })
        const client = jwksClient({ jwksUri, requestAgent });
        let publicKey = '';
        publicKey = (await client.getSigningKey(kid)).getPublicKey();
        return publicKey;
    };

    private extractTokenFromRequest(request): string | null {
        // Extract the bearer token from the request headers or other sources
        const authHeader = request.headers.authorization;
        if (authHeader && authHeader.startsWith('Bearer ')) {
            return authHeader.split(' ')[1];
        }
        return null;
    }

}
```

customRequest.ts

```
import { Request } from 'express';

export interface CustomRequest extends Request {
    userId?: string;
    role?: string;
}
```

roles.decorator.ts

```
import { SetMetadata } from "@nestjs/common";
```

```
export const Roles = (...roles: string[]) => SetMetadata('roles', roles)
```

scopes.decorator.ts

```
import { SetMetadata } from "@nestjs/common";

export const Scopes = (...scopes: string[]) => SetMetadata('scopes', scopes)
```

applicationForSettlement.controller.ts

```
import { Controller, Post, Body, Get, Param, UseGuards, Req, ForbiddenException }
from '@nestjs/common';
import { IApplicationForSettlementService } from
'../services/applicationForSettlement.service';
import { ApplicationForSettlement } from
'../models/applicationForSettlement.model';
import { Roles } from '../../../auth/roles.decorator';
import { BearerTokenAuthGuard } from '../../../auth/bearer-token-auth-guard';
import { CustomRequest } from '../../../auth/customRequest';
import { Scopes } from '../../../auth/scopes.decorator';

@Controller('')
export class ApplicationForSettlementController {
    constructor(private readonly applicationForSettlementService:
IApplicationForSettlementService) { }

    @Post('applicationForSettlement')
    @UseGuards(BearerTokenAuthGuard)
    @Roles('student')
    async add(@Req() request: CustomRequest, @Body()
applicationForSettlementCreateRequest: any): Promise<string> {
        await this.applicationForSettlementService.add(request.userId,
            applicationForSettlementCreateRequest.startYear);
        return 'ok';
    }


    @UseGuards(BearerTokenAuthGuard)
    @Roles('comendant', 'dean')
    @Get('applicationForSettlement/sign/:id')
    async sign(@Req() request: CustomRequest, @Param('id') id: string):
Promise<string> {
        await this.applicationForSettlementService.sign(id, request.userId,
request.role);
        return 'ok'
    }

    @UseGuards(BearerTokenAuthGuard)
    @Roles('comendant', 'dean')
    @Get('applicationForSettlement/studentId/:studentId')
    async getByStudentId(@Param('studentId') studentId: string):
```

```
Promise<ApplicationForSettlement[]> {
        return await
this.applicationForSettlementService.getByStudentId(studentId);
    }

    @UseGuards(BearerTokenAuthGuard)
    @Roles('student')
    @Get('applicationForSettlement/studentId')
    async getByStudent(@Req() request: CustomRequest):
Promise<ApplicationForSettlement[]> {
        return await
this.applicationForSettlementService.getByStudentId(request.userId);
    }

    @UseGuards(BearerTokenAuthGuard)
    @Roles('comendant', 'dean', 'student')
    @Scopes('api1.read')
    @Get('applicationForSettlement/:id')
    async getById(@Req() request: CustomRequest, @Param('id') id: string):
Promise<ApplicationForSettlement> {

        const applicationForSettlement = await
this.applicationForSettlementService.getById(id);

        if (request.role === 'student') {
            if (request.userId !== applicationForSettlement.studentId) {
                throw new ForbiddenException('Forbidden');
            }
        }
        return applicationForSettlement;
    }
}
```

applicationForSettlement.entity.ts

```
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import { Document } from 'mongoose';

@Schema({ collection: 'applicationForSettlement' }) // Set the custom collection
name here
export class ApplicationForSettlementDocument extends Document {

    @Prop()
    StudentId: string;

    @Prop()
    SignatoryIdDictionary: Object;

    @Prop()
    Date: string;
```

```
    @Prop()
    StartYear: number;

    @Prop()
    EndYear: number;

}

export const ApplicationForSettlementSchema =
SchemaFactory.createForClass(ApplicationForSettlementDocument);
```

applicationForSettlement.mapper.ts

```
import { ApplicationForSettlementDocument } from
"../entitys/applicationForSettlement.entity";
import { ApplicationForSettlement } from
"../../../models/applicationForSettlement.model";
import { Injectable } from "@nestjs/common";

@Injectable()
export class ApplicationForSettlementMapper {
    public mapToEntity(applicationForSettlement: ApplicationForSettlement): any {
        const applicationForSettlementDocument: any = {}
        applicationForSettlementDocument.StudentId =
applicationForSettlement.studentId;
        applicationForSettlementDocument.SignatoryIdDictionary =
applicationForSettlement.signatoryIdDictionary;
        applicationForSettlementDocument.Date = applicationForSettlement.date;
        applicationForSettlementDocument.StartYear =
applicationForSettlement.startYear;
        applicationForSettlementDocument.EndYear =
applicationForSettlement.endYear;
        return applicationForSettlementDocument;
    }

    public mapToModel(applicationForSettlementDocument:
ApplicationForSettlementDocument): ApplicationForSettlement {
        const applicationForSettlement = new ApplicationForSettlement();
        applicationForSettlement.studentId =
applicationForSettlementDocument.StudentId;
        if (applicationForSettlementDocument.SignatoryIdDictionary !== undefined) {
            applicationForSettlement.signatoryIdDictionary =
applicationForSettlementDocument.SignatoryIdDictionary;
        } else {
            applicationForSettlement.signatoryIdDictionary = {};
        }

        applicationForSettlement.date = applicationForSettlementDocument.Date;
```

```
        applicationForSettlement.startYear =
applicationForSettlementDocument.StartYear;
        applicationForSettlement.endYear =
applicationForSettlementDocument.EndYear;
        applicationForSettlement.id =
applicationForSettlementDocument._id.toString();
        return applicationForSettlement;
    }
}
```

applicationForSettlement.repository.ts

```
import { Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { Model } from 'mongoose';
import { ApplicationForSettlementDocument } from
'../entitys/applicationForSettlement.entity';
import { ApplicationForSettlementMapper } from
'../mappers/applicationForSettlement.mapper';
import { ApplicationForSettlement } from
'../../../models/applicationForSettlement.model';
import { IApplicationForSettlementRepository } from
'../../applicationForSettlement.repository';

@Injectable()
export class MongoApplicationForSettlementRepository implements
IApplicationForSettlementRepository {
    constructor(@InjectModel(ApplicationForSettlementDocument.name)
    private applicationForSettlementModel: Model<ApplicationForSettlementDocument>,
        private applicationForSettlementMapper: ApplicationForSettlementMapper) { }

    public async add(studentId: string, yearStart: number): Promise<void> {
        const applicationForSettlementDocument = {
            StudentId: studentId,
            SignatoryIdDictionary: {},
            Date: new Date(),
            StartYear: yearStart,
            EndYear: yearStart + 1
        }

        const applicationForSettlementModel = new
this.applicationForSettlementModel(applicationForSettlementDocument);
        await applicationForSettlementModel.save();
    }

    public async sign(applicationForSettlementId: string, signatoryId: string,
role: string): Promise<void> {
        const applicationForSettlementModel = await
this.applicationForSettlementModel.findById(applicationForSettlementId).exec()
        if (applicationForSettlementModel.SignatoryIdDictionary === undefined) {
            applicationForSettlementModel.SignatoryIdDictionary = {};
```

```
        }
        applicationForSettlementModel.SignatoryIdDictionary[role] = signatoryId;

        await
this.applicationForSettlementModel.findByIdAndUpdate(applicationForSettlementId,
            { SignatoryIdDictionary:
applicationForSettlementModel.SignatoryIdDictionary }, { new: true }).exec();
    }

    public async getByStudentId(studentId: string):
Promise<ApplicationForSettlement[]> {
        return (await this.applicationForSettlementModel.find({ StudentId:
studentId })
            .exec()).map(m => this.applicationForSettlementMapper.mapToModel(m))
    }

    public async getById(id: string): Promise<ApplicationForSettlement> {
        return this.applicationForSettlementMapper.mapToModel((await
this.applicationForSettlementModel.findById(id).exec()));
    }
}
```

applicationForSettlement.repository.ts

```
import { ApplicationForSettlement } from '../models/applicationForSettlement.model'

export abstract class IApplicationForSettlementRepository {
    abstract add(studentId: string, yearStart: number): Promise<void>
    abstract sign(applicationForSettlementId: string, signatoryId: string, role:
string): Promise<void>
    abstract getByStudentId(studentId: string): Promise<ApplicationForSettlement[]>
    abstract getById(applicationForSettlementId: string):
Promise<ApplicationForSettlement>
}
```

applicationForSettlement.model.ts

```
export class ApplicationForSettlement {
    id: string;
    studentId: string;
    signatoryIdDictionary: any;

    date: string;
    startYear: number;
    endYear: number;
}
```

applicationForSettlement.service.ts

```
import { ApplicationForSettlement } from '../models/applicationForSettlement.model'
```

```
export abstract class IApplicationForSettlementService {
    abstract add(studentId: string, yearStart: number): Promise<void>
    abstract sign(applicationForSettlementId: string, signatoryId: string, role:
string): Promise<void>
    abstract getByStudentId(studentId: string): Promise<ApplicationForSettlement[]>
    abstract getById(applicationForSettlementId: string):
Promise<ApplicationForSettlement>

}
```

default.applicationForSettlement.service.ts

```
import { Injectable } from '@nestjs/common';
import { IApplicationForSettlementService } from
'./applicationForSettlement.service';
import { ApplicationForSettlement } from
'../models/applicationForSettlement.model';
import { IApplicationForSettlementRepository } from
'../dal/applicationForSettlement.repository';

@Injectable()
export class DefaultApplicationForSettlementService implements
IApplicationForSettlementService {

    constructor(private applicationForSettlementRepository:
IApplicationForSettlementRepository) {

    }
    public async add(studentId: string, yearStart: number): Promise<void> {
        await this.applicationForSettlementRepository.add(studentId, yearStart);
    }

    public async sign(applicationForSettlementId: string, signatoryId: string,
role: string): Promise<void> {
        await
this.applicationForSettlementRepository.sign(applicationForSettlementId,
signatoryId, role);
    }

    public async getByStudentId(studentId: string):
Promise<ApplicationForSettlement[]> {
        return await
this.applicationForSettlementRepository.getByStudentId(studentId);
    }

    public async getById(applicationForSettlementId: string):
Promise<ApplicationForSettlement> {
        return await
this.applicationForSettlementRepository.getById(applicationForSettlementId);
    }
```

```
}
```

applicationForSettlement.module.ts

```
import { Module } from '@nestjs/common';

import { MongooseModule } from '@nestjs/mongoose';
import { ApplicationForSettlementDocument, ApplicationForSettlementSchema } from
'./dal/mongo/entitys/applicationForSettlement.entity'
import { ApplicationForSettlementController } from
'./controllers/applicationForSettlement.controller';
import { IApplicationForSettlementService } from
'./services/applicationForSettlement.service';
import { DefaultApplicationForSettlementService } from
'./services/default.applicationForSettlement.service';
import { ApplicationForSettlementMapper } from
'./dal/mongo/mappers/applicationForSettlement.mapper';
import { IApplicationForSettlementRepository } from
'./dal/applicationForSettlement.repository'
import { MongoApplicationForSettlementRepository } from
'./dal/mongo/repositories/applicationForSettlement.repository';
import { AuthModule } from '../../auth/auth.module';

@Module({
    imports: [MongooseModule.forFeature([{ name:
ApplicationForSettlementDocument.name, schema: ApplicationForSettlementSchema }]),
        AuthModule],
    controllers: [ApplicationForSettlementController],
    providers: [
        ApplicationForSettlementMapper,
        {
            provide: IApplicationForSettlementRepository,
            useClass: MongoApplicationForSettlementRepository,
        },
        {
            provide: IApplicationForSettlementService,
            useClass: DefaultApplicationForSettlementService,
        },
    ],
})
export class ApplicationForSettlementModule { }
```

debt.controller.ts

```
import { Controller, Get, Param, UseGuards, Req, ForbiddenException } from
'@nestjs/common';
import { IDebtService } from '../services/debt.service';
import { Roles } from '../../../auth/roles.decorator';
import { BearerTokenAuthGuard } from '../../../auth/bearer-token-auth-guard';
```

```
import { CustomRequest } from '../../../auth/customRequest';
import { Debt } from '../models/debt.model';


@Controller('')
export class DebtController {
    constructor(private readonly debtService: IDebtService) { }


    @Get('debt/:userId')
    @UseGuards(BearerTokenAuthGuard)
    @Roles('comendant', 'student', 'dean', 'passportHolder')
    async getDebt(@Req() request: CustomRequest, @Param('userId') userId: string):
Promise<Debt> {

        if (request.role === 'student' && request.userId !== userId) {
            throw new ForbiddenException('Forbidden');
        }

        return await this.debtService.calculateDebt(userId)
    }

}
```

debt.model.ts

```
export class Debt {
    userId: string;
    value: number;
}
```

debt.service.ts

```
import { Debt } from '../models/debt.model'


export abstract class IDebtService {
    abstract calculateDebt(userId: string): Promise<Debt>
}
```

default.debt.service.ts

```
import { Injectable } from '@nestjs/common';
import { IDebtService } from './debt.service'
import { Debt } from '../models/debt.model';
import { IMonthPriceService } from '../../monthPrice/services/monthPrice.service';
import { IRentHistoryService } from
'../../rentHistory/services/rentHistory.service';
import { IPaymentService } from '../../payment/services/payment.service';
import { RentHistory } from '@app/features/rentHistory/models/rentHistory.model';

@Injectable()
export class DefaultDebtService implements IDebtService {
```

```
    constructor(
        private monthPriceService: IMonthPriceService,
        private rentHistoryService: IRentHistoryService,
        private paymentService: IPaymentService
    ) {

    }

    public async calculateDebt(userId: string): Promise<Debt> {
        const rentHistories: RentHistory[] = await
this.rentHistoryService.getRentHistoryByUserId(userId);
        const rentHisotryMonths = this.getMonthsForRentHistories(rentHistories);
        const monthPrice: number = await
this.monthPriceService.getPriceByMonths(rentHisotryMonths);
        const paymentValues = await
this.paymentService.findPaymentValuesByUserId(userId);
        const paymentSum = this.getPaymentSum(paymentValues);
        return {
            value: monthPrice - paymentSum,
            userId: userId
        };
    }

    private getPaymentSum(paymentValues: number[]): number {
        let paymentValueSum: number = 0;
        paymentValues.forEach(paymentValue => {
            paymentValueSum += paymentValue;
        });
        return paymentValueSum;
    }

    private getMonthsForRentHistories(rentHistories: RentHistory[]): Date[] {
        const rentHistoryDictionary = {};

        rentHistories.forEach((rentHistory) => {
            const historyDates: Date[] = this.getMonthsForRentHistory(rentHistory);
            historyDates.forEach((date) => {
                rentHistoryDictionary[`${date.getFullYear()}-${date.getMonth()}`] =
date;
            });
        });
        const dates: Date[] = Object.values(rentHistoryDictionary);
        return dates;
    }

    private getMonthsForRentHistory(rentHistory: RentHistory) {
        const months = [];

        let currentDate = new Date(rentHistory.startRentDate)
        let endDate = new Date()
```

```
        if (rentHistory.isActive === false) {
            endDate = rentHistory.endRentDate
        } else {
            endDate = new Date();
            endDate.setDate(1);
            endDate.setHours(0);
            endDate.setMilliseconds(0);
            endDate.setMinutes(0);
            endDate.setSeconds(0);
        }

        // Iterate through the months until we reach or exceed the end date
        while (currentDate <= endDate) {
            months.push(new Date(currentDate)); // Clone the date to avoid
modifying the original
            // Move to the next month
            currentDate.setMonth(currentDate.getMonth() + 1);
        }

        return months;
    }
}
```

debt.module.ts

```
import { Module } from '@nestjs/common';
import { DebtController } from './controllers/debt.controller';
import { IDebtService } from './services/debt.service';
import { DefaultDebtService } from './services/default.debt.service';
import { AuthModule } from '../../auth/auth.module';
import { RentHistoryModule } from '../rentHistory/rentHistory.module';
import { MonthPriceModule } from '../monthPrice/monthPrice.module';
import { PaymentModule } from '../payment/payment.module';

@Module({
    imports: [
        AuthModule,
        RentHistoryModule,
        MonthPriceModule,
        PaymentModule],
    controllers: [DebtController],
    providers: [
        {
            provide: IDebtService,
            useClass: DefaultDebtService,
        },
    ],
})
export class DebtModule { }
```

payment.controller.ts

```
import { Controller, Get, UseGuards, Req, Param, Post, UseInterceptors,
UploadedFile, Body, Put } from '@nestjs/common';
import { IPaymentService } from '../services/payment.service';
import { Payment } from '../models/payment.model';
import { Roles } from '../../../auth/roles.decorator';
import { BearerTokenAuthGuard } from '../../../auth/bearer-token-auth-guard';
import { CustomRequest } from '../../../auth/customRequest';
import { FileInterceptor } from '@nestjs/platform-express';
import * as fs from 'fs-extra';
import { format } from 'date-fns';

@Controller('')
export class PaymentController {
    constructor(private readonly paymentService: IPaymentService) { }

    @Get('/payment')
    @UseGuards(BearerTokenAuthGuard)
    @Roles('student', 'admin', 'comendant', 'dean', 'passportHolder')
    async findSelfPayment(@Req() request: CustomRequest): Promise<Payment[]> {
        const selfUserId = request.userId;
        return await this.paymentService.findByUserId(selfUserId);
    }

    @Get('/payment/:userId')
    @UseGuards(BearerTokenAuthGuard)
    @Roles('admin', 'comendant', 'dean', 'passportHolder')
    async findPaymentByUserId(@Param('userId') userId: string): Promise<Payment[]>
{
        return await this.paymentService.findByUserId(userId);
    }

    @Put('/payment/:userId')
    @UseGuards(BearerTokenAuthGuard)
    @Roles('admin', 'comendant', 'dean', 'passportHolder')
    async updatePayment(@Param('userId') userId: string, @Body() paymentRequest:
Payment): Promise<string> {
        await this.paymentService.update(paymentRequest);
        return 'ok';
    }

    @Post('/payment')
    @UseGuards(BearerTokenAuthGuard)
    @Roles('student')
    @UseInterceptors(FileInterceptor('file'))
    async updateImage(@Req() request: CustomRequest,
        @UploadedFile() file: Express.Multer.File,
        @Body() paymentRequest: Payment): Promise<void> {
        const selfUserId = request.userId;
```

```
        const buffer = await fs.readFile(file.path);
        await fs.unlink(file.path);

        const payment = new Payment();
        payment.userId = selfUserId;

        const currentDate = new Date();
        payment.date = format(currentDate, 'dd.MM.yyyy HH:mm:ss');
        payment.paymentApproved = false;
        payment.paymentImage = buffer.toString('base64');

        if (paymentRequest) {
            payment.recognizedDestinationCardNumber =
paymentRequest.recognizedDestinationCardNumber;
            payment.recognizedPaymentId = paymentRequest.recognizedPaymentId;
            payment.recognizedValue = paymentRequest.recognizedValue;
        }

        await this.paymentService.add(payment);
    }



}
```

payment.entity.ts

```
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import { Document } from 'mongoose';

@Schema({ collection: 'Payment' }) // Set the custom collection name here
export class PaymentDocument extends Document {
    @Prop()
    UserId: string;

    @Prop()
    PaymentImage: Buffer;

    @Prop()
    Date: string;

    @Prop()
    Value: number | null;

    @Prop()
    PaymentApproved: boolean | null;

    @Prop()
    PaymentId: string | null;

    @Prop()
```

```
    PaymentDestinationCardNumber: string | null;

    @Prop()
    RecognizedValue: number | null;

    @Prop()
    RecognizedPaymentId: string | null;

    @Prop()
    RecognizedDestinationCardNumber: string | null;

}

export const PaymentSchema = SchemaFactory.createForClass(PaymentDocument);
```

payment.mapper.ts

```
import { PaymentDocument } from "../entitys/payment.entity";
import { Payment } from "../../../models/payment.model";
import { Injectable } from "@nestjs/common";

@Injectable()
export class PaymentMapper {

    public mapToEntity(payment: Payment): any {
        const paymentDocument: any = {}
        paymentDocument.UserId = payment.userId;
        paymentDocument.PaymentImage = Buffer.from(payment.paymentImage,
'base64');;
        paymentDocument.Date = payment.date;
        paymentDocument.Value = payment.value;
        paymentDocument.PaymentApproved = payment.paymentApproved;
        paymentDocument.PaymentId = payment.paymentId;
        paymentDocument.PaymentDestinationCardNumber =
payment.paymentDestinationCardNumber;
        paymentDocument.RecognizedDestinationCardNumber =
payment.recognizedDestinationCardNumber;
        paymentDocument.RecognizedPaymentId = payment.recognizedPaymentId;
        paymentDocument.RecognizedValue = payment.recognizedValue;
        return paymentDocument;
    }

    public mapToModel(paymentDocument: PaymentDocument): Payment {
        const payment = new Payment();
        payment.id = paymentDocument._id.toString();
        payment.paymentImage = paymentDocument.PaymentImage.toString('base64');
        payment.date = paymentDocument.Date;
        payment.userId = paymentDocument.UserId;
        payment.paymentApproved = paymentDocument.PaymentApproved;
        payment.value = paymentDocument.Value;
```

```
        payment.paymentId = paymentDocument.PaymentId;
        payment.paymentDestinationCardNumber =
paymentDocument.PaymentDestinationCardNumber;
        payment.recognizedDestinationCardNumber =
paymentDocument.RecognizedDestinationCardNumber;
        payment.recognizedPaymentId = paymentDocument.RecognizedPaymentId;
        payment.recognizedValue = paymentDocument.RecognizedValue;
        return payment;
    }
}
```

payment.repository.ts

```
import { Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { Model } from 'mongoose';
import { PaymentDocument } from '../entitys/payment.entity';
import { PaymentMapper } from '../mappers/payment.mapper';
import { Payment } from '../../../models/payment.model';
import { IPaymentRepository } from '../../payment.repository';

@Injectable()
export class MongoPaymentRepository implements IPaymentRepository {
    constructor(@InjectModel(PaymentDocument.name) private paymentModel:
Model<PaymentDocument>,
        private paymentMapper: PaymentMapper) { }

    public async findVerifiedPaymentValuesByUserId(userId: string):
Promise<number[]> {
        const paymentValues = await this.paymentModel.aggregate([
            { $match: { UserId: userId, PaymentApproved: true } }, // Filter
documents
            //{ $project: { Value: 1, _id: 0 } }, // Project only the specified
field
        ]).exec();

        const payments: number[] = [];
        paymentValues.forEach((paymentValue: PaymentDocument) => {
            if (paymentValue.Value) {
                payments.push(paymentValue.Value);
            } else {
                if (paymentValue.RecognizedValue) {
                    payments.push(paymentValue.RecognizedValue);
                }
            }
        })
        return payments;
    }

    async findAll(): Promise<Payment[]> {
        return (await this.paymentModel.find().exec()).map(n =>
```

```
this.paymentMapper.mapToModel(n));
    }

    async findByUserId(userId: string): Promise<Payment[]> {
        return (await this.paymentModel.find({ 'UserId': userId }).exec()).map(m =>
this.paymentMapper.mapToModel(m));
    }

    async add(payment: Payment): Promise<void> {
        const paymentDocument = this.paymentMapper.mapToEntity(payment);
        const paymentModel = new this.paymentModel(paymentDocument);
        await paymentModel.save();
    }

    async update(payment: Payment): Promise<void> {

        const paymentEntity = {
            Value: payment.value,
            PaymentId: payment.paymentId,
            PaymentDestinationCardNumber: payment.paymentDestinationCardNumber,
            PaymentApproved: payment.paymentApproved
        }

        await this.paymentModel.findByIdAndUpdate(payment.id, paymentEntity, { new:
true }).exec();
    }

}
```

payment.repository.ts

```
import { Payment } from '../models/payment.model'

export abstract class IPaymentRepository {
    abstract findAll(): Promise<Payment[]>

    abstract findByUserId(userId: string): Promise<Payment[]>

    abstract add(payment: Payment): Promise<void>

    abstract findVerifiedPaymentValuesByUserId(userId: string): Promise<number[]>

    abstract update(payment: Payment): Promise<void>
}
```

payment.model.ts

```
export class Payment {
    id: string;
    userId: string;
    paymentImage: string;
```

```
    date: string;

    value: number | null;
    paymentId: string | null;
    paymentDestinationCardNumber: string | null;

    recognizedValue: number | null;
    recognizedPaymentId: string | null;
    recognizedDestinationCardNumber: string | null;

    paymentApproved: boolean | null;
}
```

default.payment.service.ts

```
import { Injectable } from '@nestjs/common';
import { IPaymentService } from './payment.service'
import { Payment } from '../models/payment.model';
import { IPaymentRepository } from '../dal/payment.repository';

@Injectable()
export class DefaultPaymentService implements IPaymentService {

    constructor(private paymentRepository: IPaymentRepository) {

    }

    public async findPaymentValuesByUserId(userId: string): Promise<number[]> {
        return await
this.paymentRepository.findVerifiedPaymentValuesByUserId(userId);
    }

    async findAll(): Promise<Payment[]> {
        return await this.paymentRepository.findAll();
    }

    async findByUserId(userId: string): Promise<Payment[]> {
        return await this.paymentRepository.findByUserId(userId);
    }

    async add(payment: Payment): Promise<void> {
        await this.paymentRepository.add(payment);
    }

    async update(payment: Payment): Promise<void> {
        await this.paymentRepository.update(payment);
    }
}
```

payment.service.ts

```
import { Payment } from '../models/payment.model'
```

```
export abstract class IPaymentService {
    abstract findAll(): Promise<Payment[]>

    abstract findByUserId(userId: string): Promise<Payment[]>

    abstract add(payment: Payment): Promise<void>

    abstract findPaymentValuesByUserId(userId: string): Promise<number[]>

    abstract update(payment: Payment): Promise<void>
}
```

payment.module.ts

```
import { Module } from '@nestjs/common';

import { MongooseModule } from '@nestjs/mongoose';
import { PaymentDocument, PaymentSchema } from './dal/mongo/entitys/payment.entity'
import { PaymentController } from './controllers/payment.controller';
import { IPaymentService } from './services/payment.service';
import { DefaultPaymentService } from './services/default.payment.service';
import { PaymentMapper } from './dal/mongo/mappers/payment.mapper';
import { IPaymentRepository } from './dal/payment.repository'
import { MongoPaymentRepository } from
'./dal/mongo/repositories/payment.repository'
import { AuthModule } from '../../auth/auth.module';
import { MulterModule } from '@nestjs/platform-express/multer';
import { multerConfig } from '../../multer/multer.config'

@Module({
    imports: [
        MulterModule.register(multerConfig),
        MongooseModule.forFeature([{ name: PaymentDocument.name, schema:
PaymentSchema }]),
        AuthModule],
    controllers: [PaymentController],
    providers: [
        PaymentMapper,
        {
            provide: IPaymentRepository,
            useClass: MongoPaymentRepository,
        },
        {
            provide: IPaymentService,
            useClass: DefaultPaymentService,
        },
    ],
    exports: [IPaymentService]
})
export class PaymentModule { }
```

ImageRecognitionController.cs

```
using IdentityProvider.Features.PaymentCheckRecognition.Models;
using IdentityProvider.Features.PaymentCheckRecognition.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Data;
using Tesseract;
using static Duende.IdentityServer.IdentityServerConstants;

namespace IdentityProvider.Controllers
{
  [Authorize(Policy = LocalApi.PolicyName, Roles = "student,admin,comendant,dean,passportHolder")]
  public class ImageRecognitionController : ApiController
  {
    private readonly IPaymentCheckRecognitionService _paymentCheckRecognitionService;
    public ImageRecognitionController(IPaymentCheckRecognitionService paymentCheckRecognitionService) {
      _paymentCheckRecognitionService = paymentCheckRecognitionService;
    }

    [HttpPost]
    [Route("/recognize/paymentCheck")]
    public CheckPaymentInfo RecognizePaymentCheck()
    {
      return _paymentCheckRecognitionService.RecognizeCheckPaymentInfo(GetImageFromRequestBytes());
    }
  }
}
```

PaymentCheckRecognitionService.cs

```
using IdentityProvider.Features.PaymentCheckRecognition.Models;
using IdentityProvider.Features.PaymentCheckRecognition.Services.PaymentCheckRecognitionStrategies;
using System.Collections.Generic;
using System.Drawing;
using Tesseract;

namespace IdentityProvider.Features.PaymentCheckRecognition.Services
{
  public class PaymentCheckRecognitionService : IPaymentCheckRecognitionService
  {
    private List<IPaymentCheckStrategy> paymentCheckStrategies = new List<IPaymentCheckStrategy>()
    {
      new PrivatBankDesktopPaymentCheckStrategy()
    };

    public CheckPaymentInfo RecognizeCheckPaymentInfo(byte[] image)
    {
      CheckPaymentInfo payemtnInfoResult = null;
      int maxRecognitions = -1;
      foreach (var recognitionStartegy in paymentCheckStrategies)
      {
        int recognizeResult = recognitionStartegy.Recognize(image, out CheckPaymentInfo payemtnInfo);
        if (recognizeResult > maxRecognitions)
        {
          payemtnInfoResult = payemtnInfo;
          maxRecognitions = recognizeResult;
        }
      }
```

```
            return payemtnInfoResult;

        }
    }
}
```

IPaymentCheckRecognitionService.cs

```
using IdentityProvider.Features.PaymentCheckRecognition.Models;

namespace IdentityProvider.Features.PaymentCheckRecognition.Services
{
    public interface IPaymentCheckRecognitionService
    {
        CheckPaymentInfo RecognizeCheckPaymentInfo(byte[] image);
    }
}
```

IPaymentCheckStrategy.cs

```
using IdentityProvider.Features.PaymentCheckRecognition.Models;

namespace IdentityProvider.Features.PaymentCheckRecognition.Services.PaymentCheckRecognitionStrategies
{
    public interface IPaymentCheckStrategy
    {
        int Recognize(byte[] image, out CheckPaymentInfo checkPaymentInfo);
    }
}
```

PrivatBankDesktopPaymentCheckStrategy.cs

```
using IdentityProvider.Features.PaymentCheckRecognition.Models;
using System.Text.RegularExpressions;
using System;
using Tesseract;
using System.Collections.Generic;
using System.Globalization;

namespace IdentityProvider.Features.PaymentCheckRecognition.Services.PaymentCheckRecognitionStrategies
{
    public class PrivatBankDesktopPaymentCheckStrategy : IPaymentCheckStrategy
    {
        public int Recognize(byte[] image, out CheckPaymentInfo checkPaymentInfo)
        {
            string dataDir = @"./tessdata";

            // Initialize Tesseract engine
            using (var engine = new TesseractEngine(dataDir, "eng+ukr", EngineMode.Default))
            {
                // Load an image (replace with your image path)
                using (var img = Pix.LoadFromMemory(image))
                {

                    string destinationCardNumber = GetDestinationCardNumber(engine, img);
                    string checkPaymentId = GetCheckPaymentId(engine, img);
                    double checkPaymentPrice = GetCheckPaymentPrice(engine, img);
                    checkPaymentInfo = new CheckPaymentInfo()
                    {
                        Id = checkPaymentId,
                        DestinationCardNumber = destinationCardNumber,
                        Price = checkPaymentPrice
```

```
        };

        int recognitionCount = 0;

        if (destinationCardNumber != null)
        {
          recognitionCount++;
        }

        if (checkPaymentId != null)
        {
          recognitionCount++;
        }

        if (checkPaymentPrice != null)
        {
          recognitionCount++;
        }

        return recognitionCount;
      }
    }
}

private string GetDestinationCardNumber(TesseractEngine engine, Pix img)
{
  try
  {
    Rect roi = new Rect(img.Width * 4 / 10, 1, img.Width * 6 / 10, img.Height - 2);
    using (var page = engine.Process(img, roi))
    {
      // Get the recognized text
      string recognizedText = page.GetText();

      string pattern = @"\bUA\d+\b";

      Regex regex = new Regex(pattern);

      var matches = regex.Matches(recognizedText);

      if (matches.Count == 1)
      {
        return matches[0].Value;
      }
      else
      {
        return null;
      }
    }
  }
  catch (Exception exc)
  {
    return null;
  }
}

private string GetCheckPaymentId(TesseractEngine engine, Pix img)
{
  try
  {
    Rect roi = new Rect(img.Width / 2, 1, img.Width / 2, img.Height * 3 / 10);
    using (var page = engine.Process(img, roi))
```

```
            {
                // Get the recognized text
                string recognizedText = page.GetText();

                string pattern = @"(\d{4}-\d{4}-\d{4}-\d{4})";

                Regex regex = new Regex(pattern);

                var matches = regex.Matches(recognizedText);

                if (matches.Count == 1)
                {
                    return matches[0].Value;
                }
                else
                {
                    return null;
                }
            }
        }
        catch (Exception exc)
        {
            return null;
        }
    }

    private double GetCheckPaymentPrice(TesseractEngine engine, Pix img)
    {
        try
        {
            Rect roi = new Rect(img.Width * 15 / 20, 1, img.Width * 5 / 20, img.Height - 2);
            using (var page = engine.Process(img, roi))
            {
                // Get the recognized text
                string recognizedText = page.GetText();

                string pattern = @"(?<!\d)(\d+\.\d+)(?!\d)";

                Regex regex = new Regex(pattern);

                var matches = regex.Matches(recognizedText);

                if (matches.Count == 1)
                {
                    return double.Parse(matches[0].Value, CultureInfo.InvariantCulture);
                }
                else
                {
                    return -1;
                }
            }
        }
        catch (Exception exc)
        {
            return -1;
        }
    }
  }
}
```

ДОДАТОК Г
Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МЕТОДІВ І ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПРОЦЕСУ УПРАВЛІННЯ ГУРТОЖИТКОМ УНІВЕРСИТЕТУ

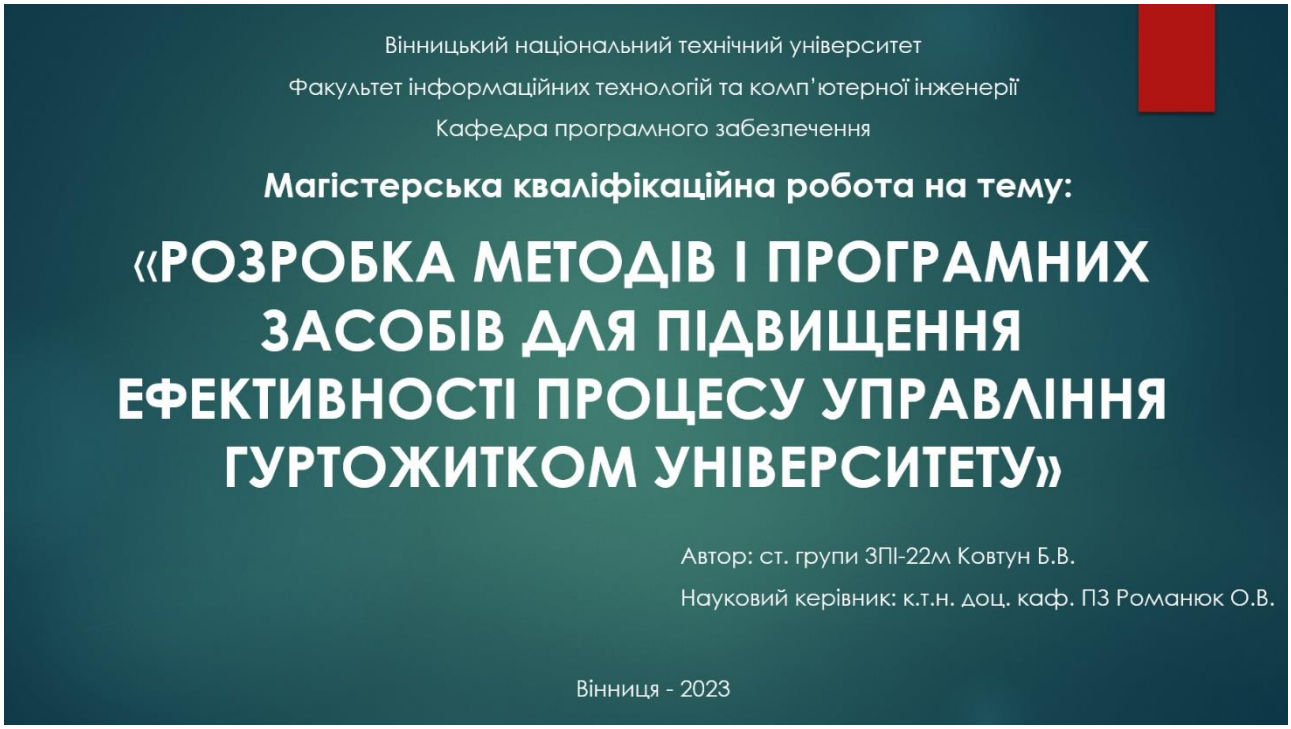


Рисунок Г.1 – Титульний слайд

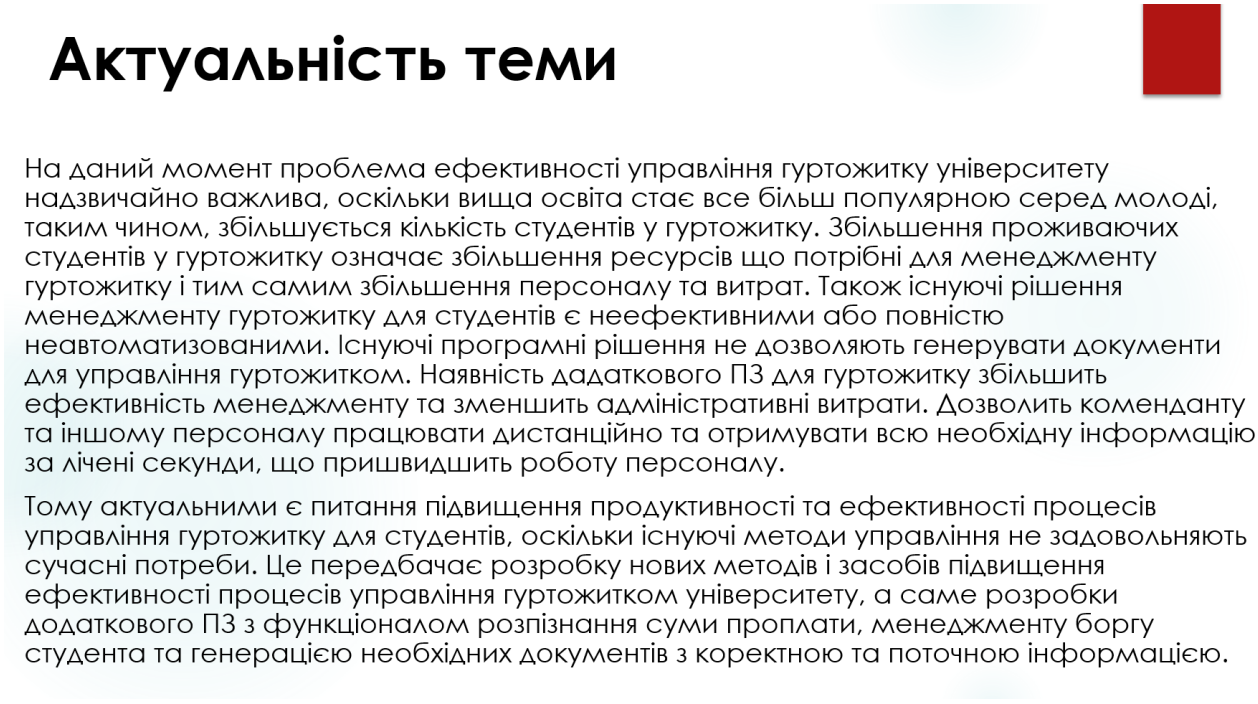


Рисунок Г.2 – Актуальність теми

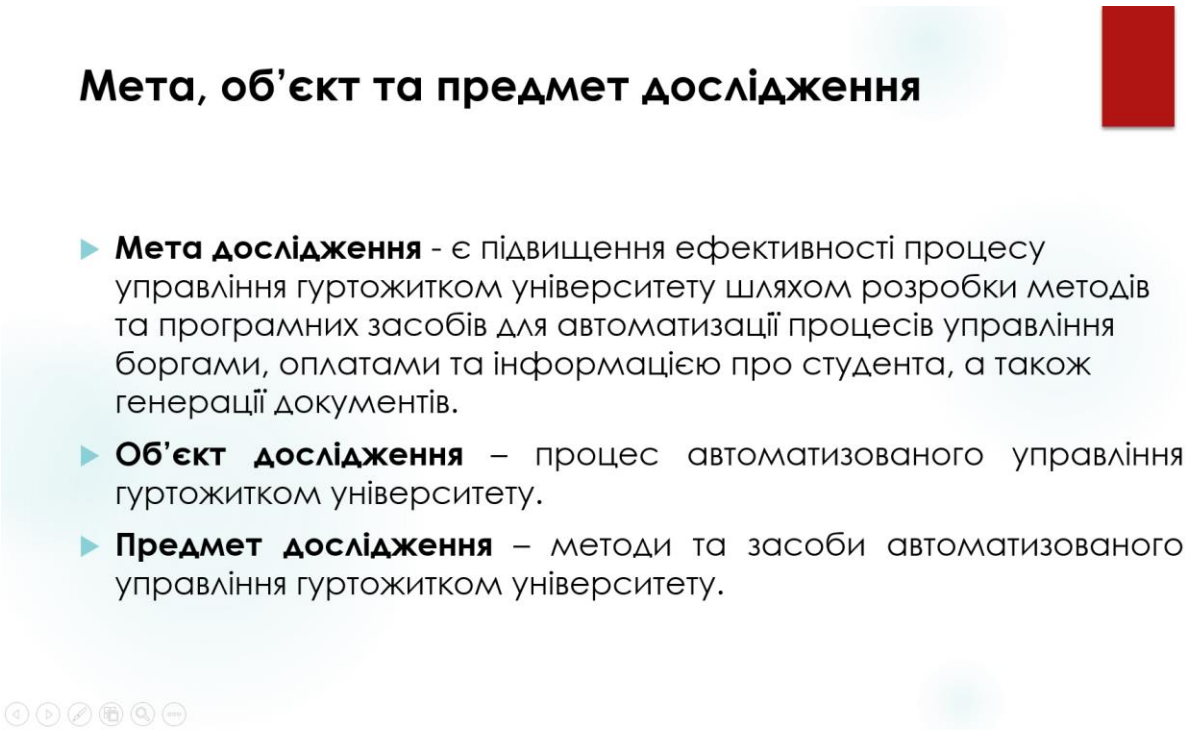


Рисунок Г.3 – Мета, об’єкт та предмет дослідження

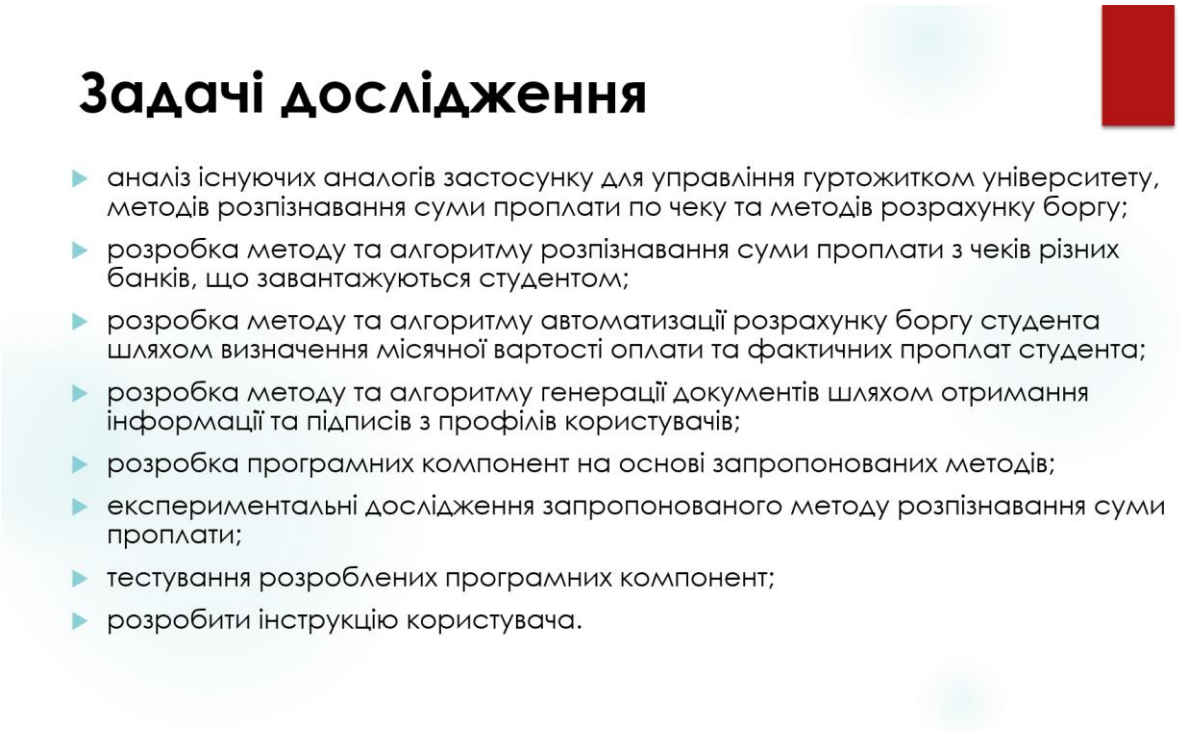


Рисунок Г.4 – Задачі дослідження

Рисунок Г.5 – Наукова новизна

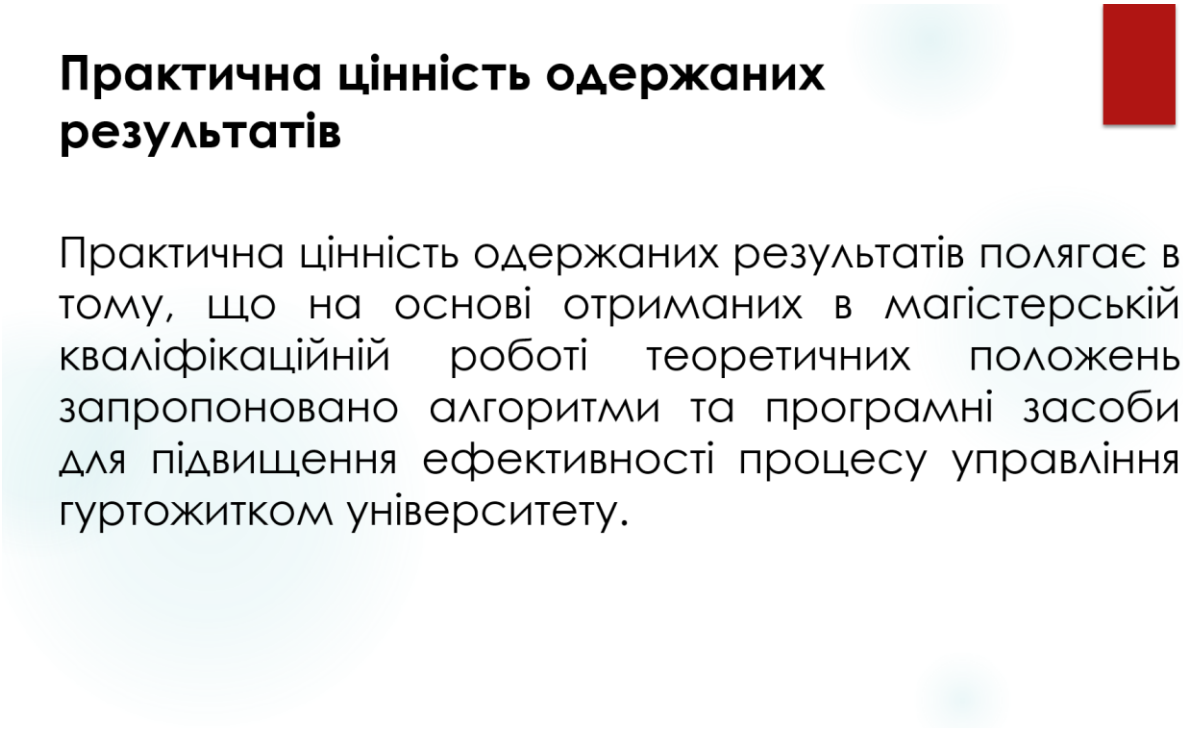


Рисунок Г.6 – Практична цінність одержаних результатів

Порівняльний аналіз аналогів

Критерій	Creatrix Campus	StarRez	Ellucian Hosing	Entrata Student	Dormament Master
Можливість валідації проплати	0	0	0	0	1
Можливість менеджменту боргу	1	1	0	1	1
Можливість генерації документів для поселення	0	1	1	1	1
Менеджмент інформації про студента	1	1	1	1	1
Підсумковий результат	2	3	2	3	4

Рисунок Г.7 – Порівняльний аналіз аналогів

Формула обчислення боргу студента

$$\text{debt} = \sum_{i=1}^{m} \text{month\_price} - \sum_{i=1}^{p} \text{approved\_payment},$$

debt – борг у гривнях, можливе мінусове число, якщо студент вніс більшу суму ніж сума місяців, що прожив студент у гуртожитку;

month_price – ціна за місяць;

m – кількість місяців, що прожив студент у гуртожитку;

approved_payment – перевірена керуючим персоналом проплата;

p – кількість перевірених проплат студента.

Рисунок Г.8 – Формула обчислення боргу студента

Метод і блок-схема алгоритму розпізнання суми проплати

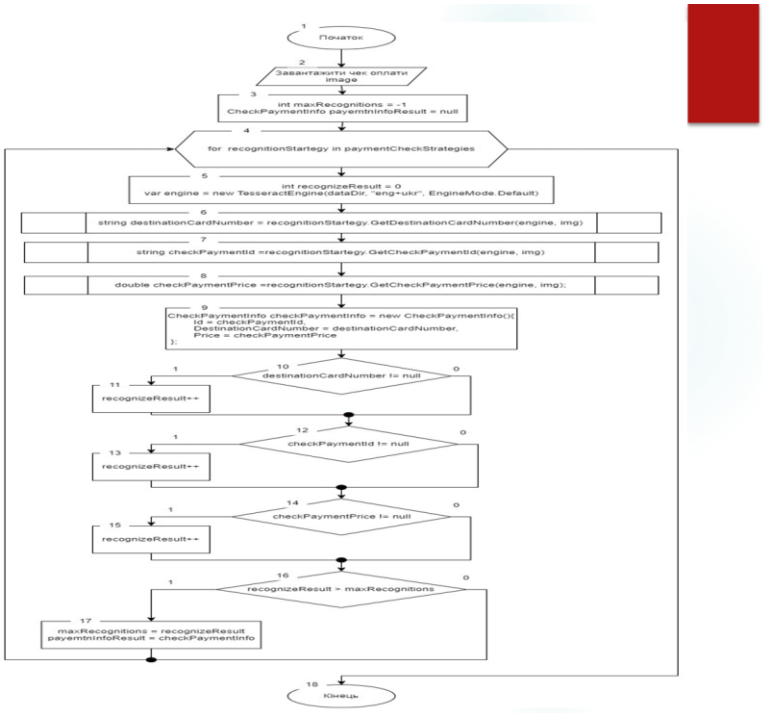


Рисунок Г.9 – Метод і блок-схема алгоритму розпізнання суми проплати

Метод і блок-схема алгоритму розрахунку боргу студента

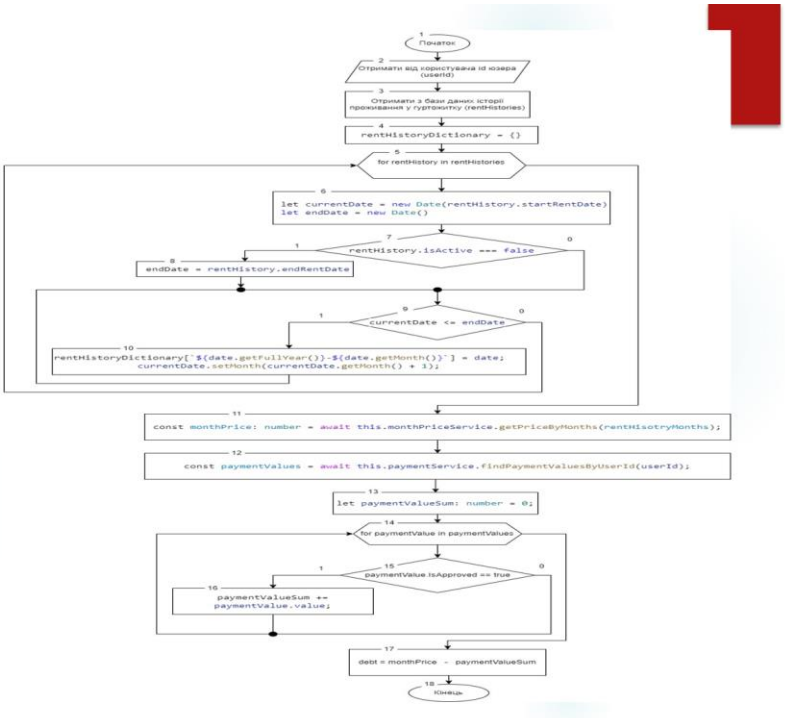


Рисунок Г.10 – Метод і блок-схема алгоритму розрахунку боргу студента

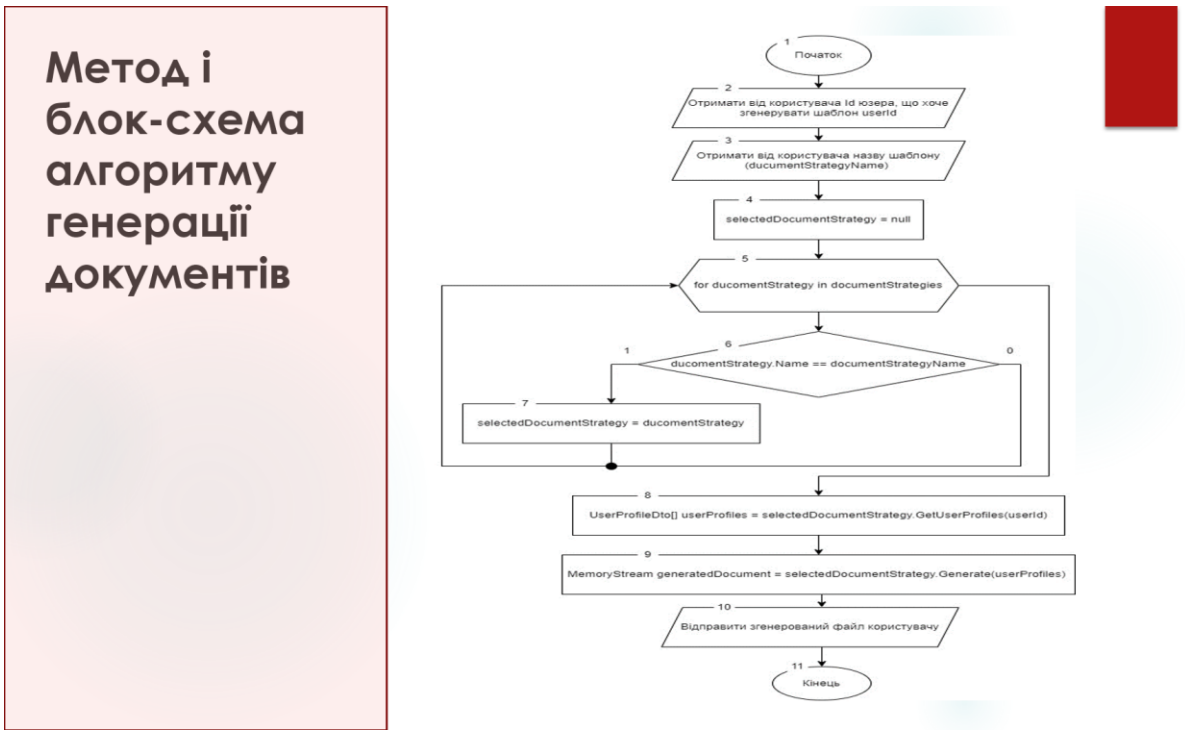


Рисунок Г.11 – Метод і блок-схема алгоритму генерації документів

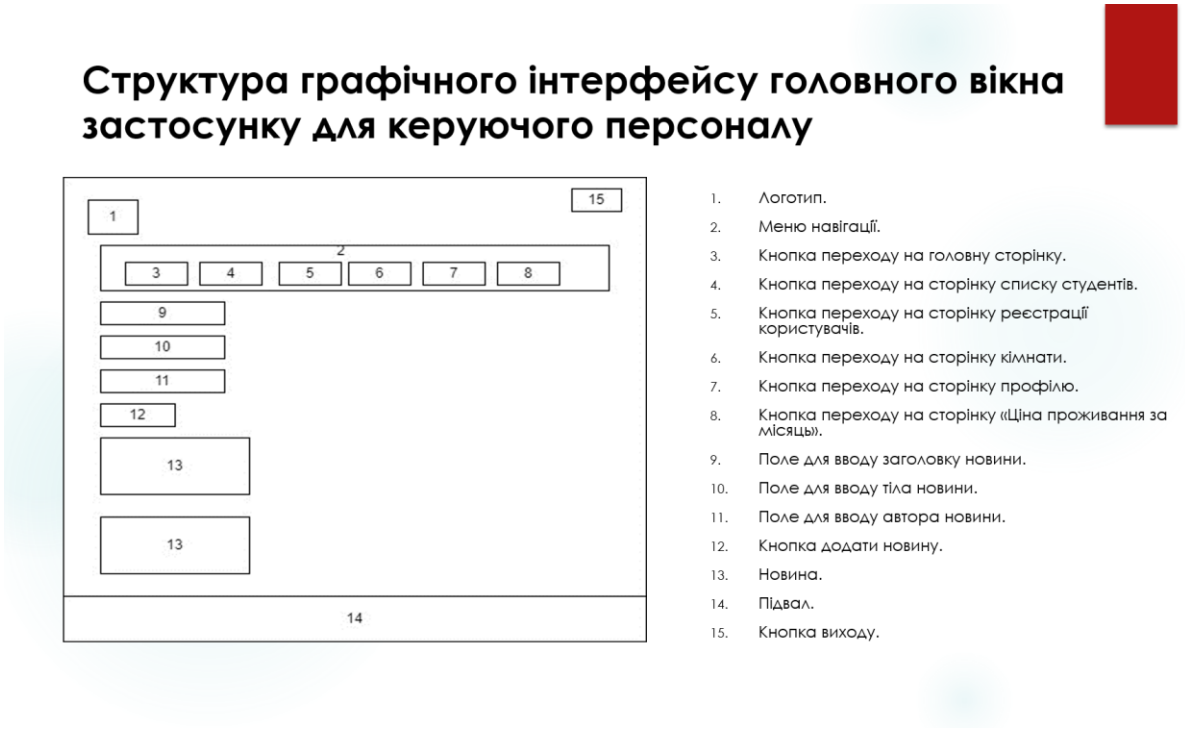


Рисунок Г.12 – Структура графічного інтерфейсу головного вікна застосунку для керуючого персоналу

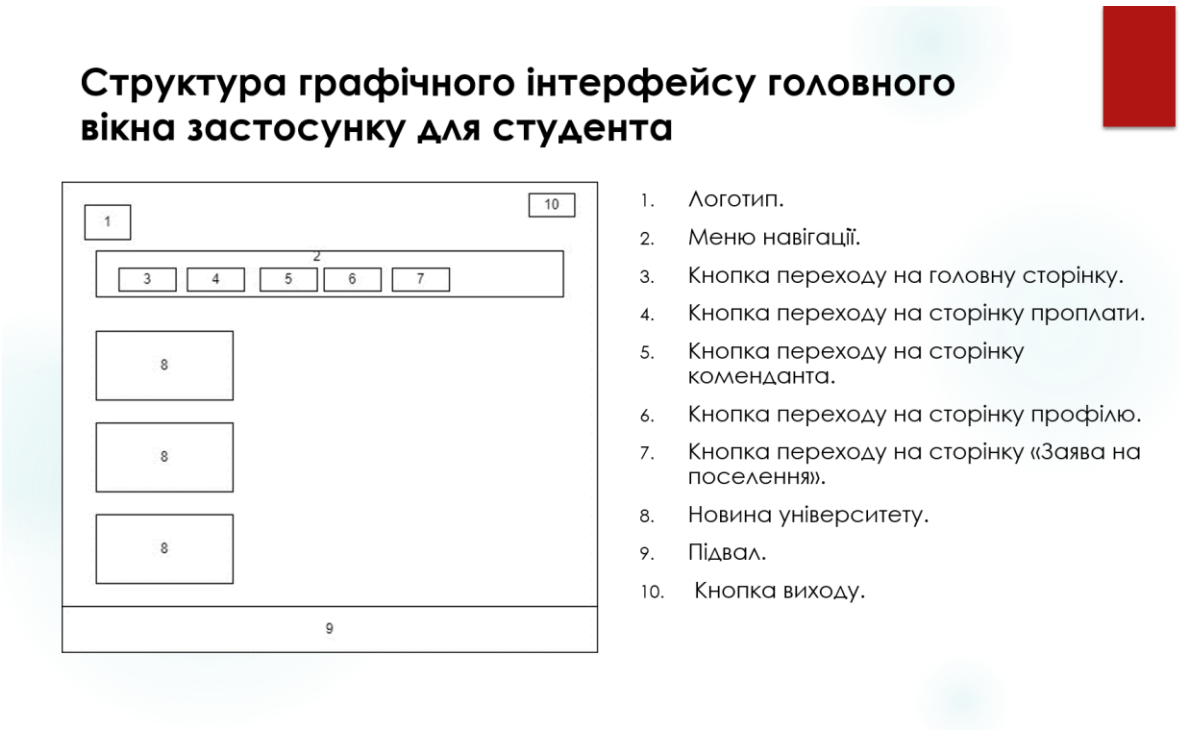


Рисунок Г.13 – Структура графічного інтерфейсу головного вікна застосунку для студента

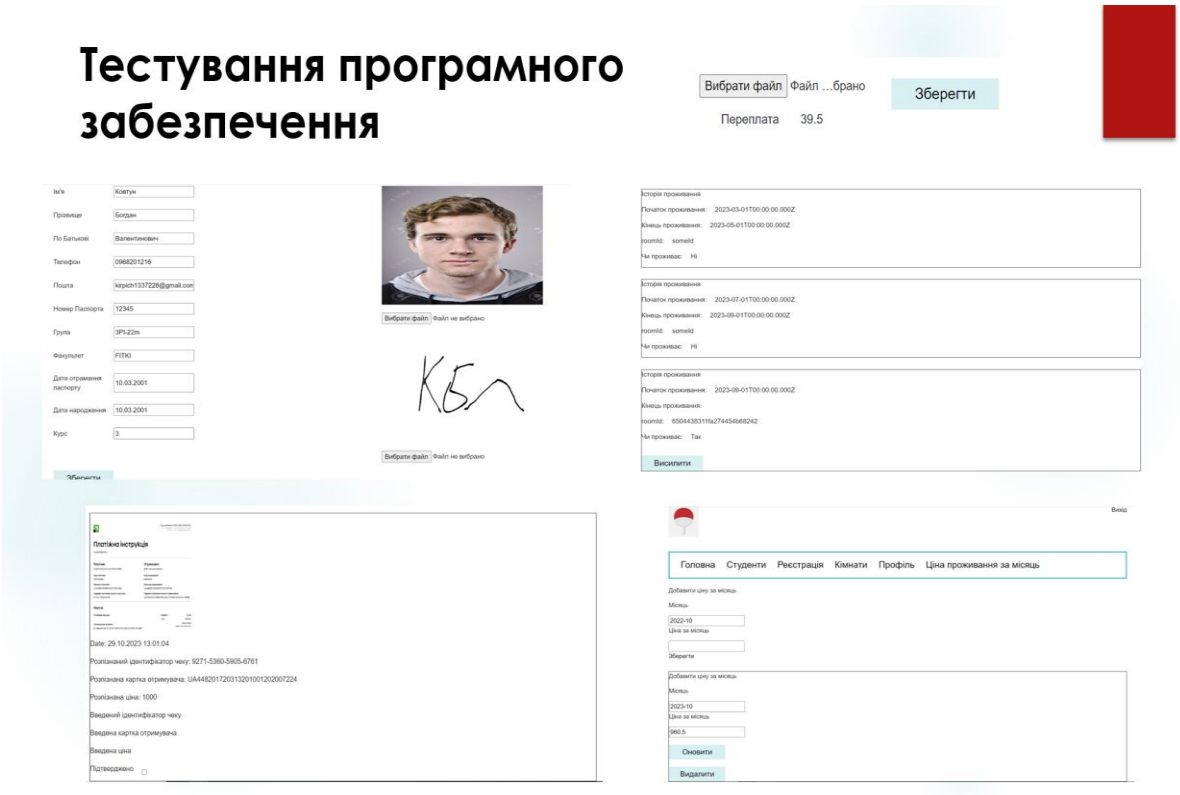


Рисунок Г.14 – Тестування програмного забезпечення

Експериментальні дослідження методу розпізнання суми проплати студента

Розпізнання	Кількість розпізнань, звичайний метод	Кількість розпізнань, розроблюваний метод
Код документу	8	10
Сума проплати	6	9
Рахунок отримувача	7	10
Всього	21	29

Як показала таблиця розроблюваний метод розпізнав 29 пунктів необхідної інформації з 30, а у той час звичайний метод розпізнання розпізнав 21 пункт. Звичайний метод розпізнав 70% необхідної інформації, а розроблюваний метод розпізнав 96%. Розроблюваний метод розпізнання проплати розпізнав інформацію на 26% точніше.

Рисунок Г.15 – Експериментальні дослідження методу розпізнання суми проплати студента

Економічна частина

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету» становить 40,7 балів, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

Також термін окупності становить 2,2 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Розробка методів і програмних засобів для підвищення ефективності процесу управління гуртожитком університету».

- Загальні витрати на проведення науково-технічної розробки та оформлення її результатів - 368133,60 грн
- Абсолютний економічний ефект вкладених інвестицій - 1519552,06 грн

Рисунок Г.16 – Економічна частина

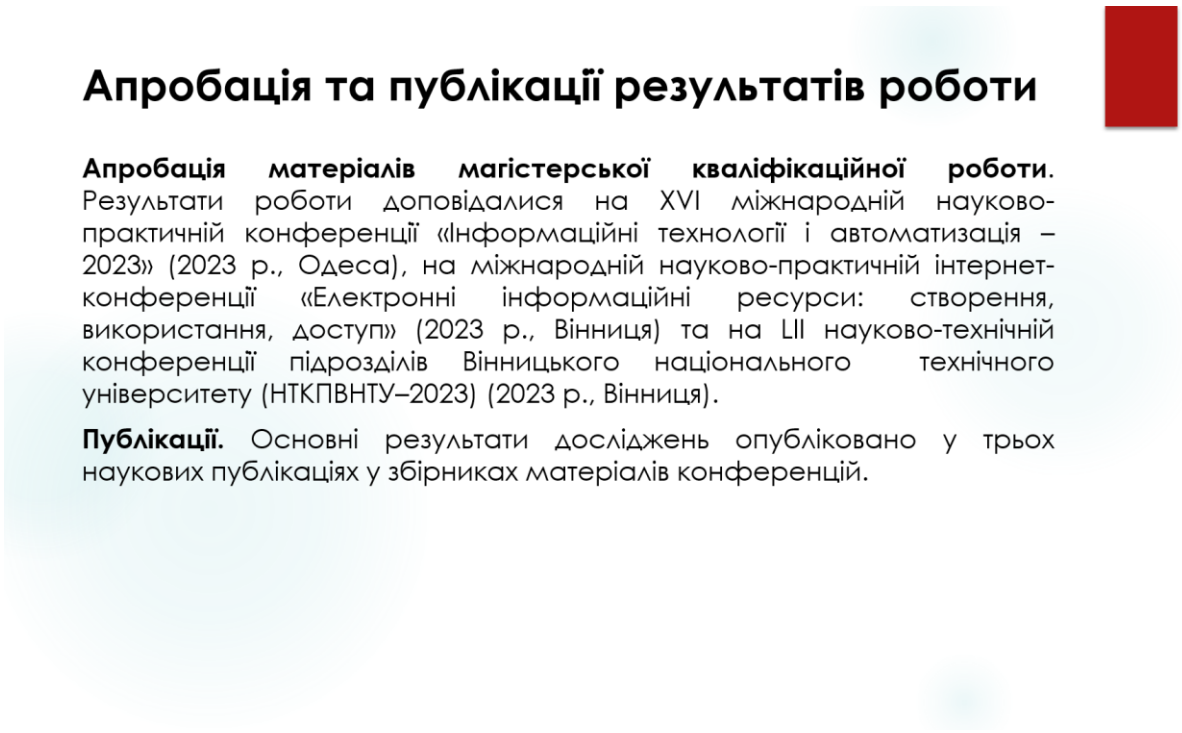


Рисунок Г.17 – Апробація та публікації результатів роботи

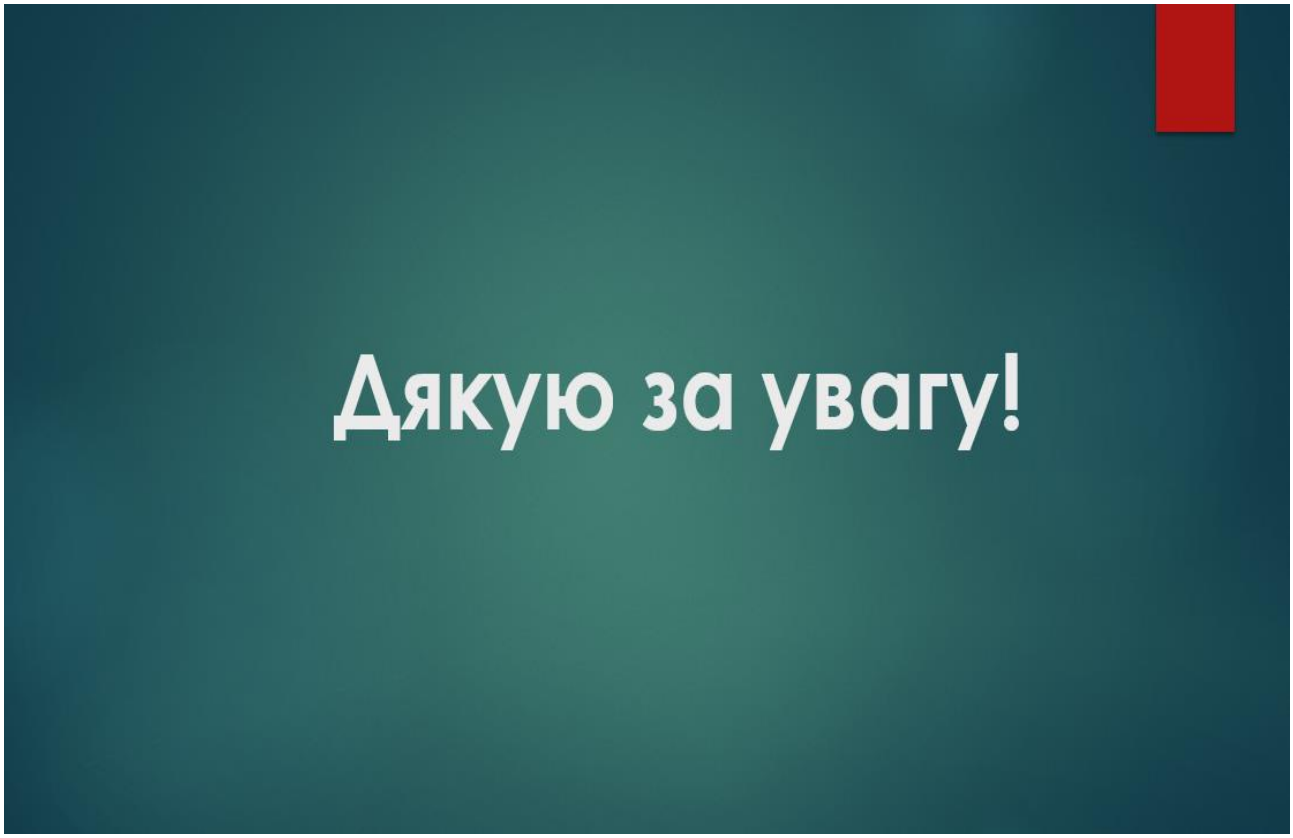


Рисунок Г.18 – Фінальний слайд