Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: Розробка програмної системи обліку персональних проїзних карток м. Вінниці

Виконав: студент 4 курсу
групи 5ПІ-21Б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Томай Андрій Олександрович
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Бабюк Н.П.
(прізвище та ініціали)

Рецензент:. к.т.н., доц. каф. ОТ Колесник І.С.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ Романюк О.Н
«09» 06 2025 р.

ВНТУ – 2025

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

ЗАТВЕРДЖУЮ
Директор КП «ВІЦ»
С. С. Бригадир
«21» березня 2025 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«21» березня 2025 р.

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

Томаю Андрію Олександровичу

1. Тема роботи – Розробка програмної системи обліку персональних проїзних карток м. Вінниці.

2. Керівник роботи: Бабюк Наталя Петрівна затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

3. Строк подання студентом роботи 2 червня 2025 р.

4. Вихідні дані до роботи: є інформація про сучасні методи автоматизованого обліку персональних проїзних карток та програмні засоби для їх реалізації. Також використовуються вимоги до безпеки, інтеграції з існуючими транспортними системами та обробки даних у реальному часі. Зміст текстової частини:

5. Перелік ілюстративного матеріалу: діаграми, схеми архітектури системи, інтерфейси користувача, таблиці з характеристиками функціональних модулів, а також графічні зображення алгоритмів обробки даних.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Бабюк Н.П., к.н.т., доцент каф. ПЗ	24.03.25р.	01.06.25р.

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану системи обліку персональних проїзних карток у сучасному світі	25.03.25 – 01.04.25	вик.
2	Аналіз аналогів програмної системи обліку персональних проїзних карток	01.04.25 – 08.04.25	вик
3	Розробка алгоритмів системи	08.04.25 – 29.04.25	вик
4	Розробка модулів системи	29.04.25 – 20.05.25	вик.
5	Тестування системи	20.05.25 – 28.05.25	вик.
6	Оформлення матеріалів до захисту БКР	28.05.25 – 01.06.25	вик.

Студент _____ **А.О.Томай**

(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ **Н.П.Бабюк**

(підпис) (ініціали та прізвище)

АНОТАЦІЯ

УДК 004.912.032.26

Томай А.О. Розробка програмної системи обліку персональних проїзних карток м. Вінниці: бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 65 с.

На укр. мові. Бібліогр. : 21 назв.; рис. : 13; табл. : 5.

У бакалаврській кваліфікаційній роботі проведено детальний аналіз автоматизованих систем обліку персональних проїзних карток. Запропоновано використання сучасних інформаційних технологій для підвищення ефективності управління транспортними системами. Розроблено інформаційну систему, що дозволяє здійснювати реєстрацію, блокування та контроль за використанням карток у режимі реального часу. Запропонована система була реалізована із використанням мов програмування Python + Flask, HTML + CSS, PyCharm як IDE, База даних PostgreSQL. Отримані результати можуть бути використані для вдосконалення міських транспортних систем шляхом автоматизації обліку проїзних карток.

Ключові слова: автоматизація, інформаційні системи, управління проїзними картками, транспортні технології.

ABSTRACT

UDK 004.912.032.26

Tomai A.O. Development of a software system for accounting for personal travel cards in Vinnytsia: bachelor's thesis in the specialty 121 Software Engineering, educational program - Software Engineering. Vinnytsia: VNTU, 2025. 65 c.

In Ukrainian. Bibliography: 21 titles; Figs. 13; Tables: 5.

In the bachelor's qualification work, a detailed analysis of automated personal travel card accounting systems is carried out. The use of modern information technologies to improve the efficiency of transport systems management is proposed. An information system has been developed that allows for real-time registration, blocking, and control over the use of cards. The proposed system was implemented using the programming languages Python + Flask, HTML + CSS, PyCharm as an IDE, and the PostgreSQL database. The obtained results can be used to improve urban transportation systems by automating the accounting of travel cards.

Keywords: automation, information systems, travel card management, transport technologies.

ЗМІСТ

ВСТУП

Розвиток інформаційних технологій значно вплинув на всі сфери життя, зокрема на транспортну систему та методи управління пасажирськими перевезеннями. Автоматизовані системи обліку проїзду дозволяють підвищити ефективність контролю за використанням проїзних карток, зменшити витрати на їхню обробку та мінімізувати можливість шахрайства. Сучасні рішення в цій сфері сприяють оптимізації транспортних потоків, покращенню сервісу для пасажирів та підвищенню рівня автоматизації процесів.

Одним із важливих елементів транспортної інфраструктури є система електронних проїзних карток, яка дозволяє громадянам зручно оплачувати проїзд у громадському транспорті. Проте існуючі системи обліку проїзних карток мають низку недоліків: недостатню інтеграцію з іншими сервісами, обмежену функціональність для користувачів, відсутність персоналізованих налаштувань та складний процес ремонту. Це створює потребу у вдосконаленні таких систем, щоб забезпечити зручний доступ до інформації та підвищити ефективність їхньої роботи.

Тому розробка програмної системи обліку персональних проїзних карток є актуальною темою. Така система має забезпечити автоматизований облік карток, автоматизацію ручної праці, можливість продовження пільги, продовження терміну дії, ремонту, блокування та розблокування карток.

Враховуючи стрімкий розвиток інформаційних технологій та зростаючу необхідність автоматизації транспортних систем, створення програмного забезпечення для обліку проїзних карток є важливим кроком у підвищенні якості громадського транспорту. Такий підхід дозволить не лише покращити досвід користувачів, а й сприятиме підвищенню загальної ефективності функціонування транспортної інфраструктури міста.

Сучасні аналоги систем обліку проїзних карток, хоча й виконують базові функції контролю, мають низку обмежень. Зокрема, вони можуть мати незручний інтерфейс, що ускладнює навігацію для користувачів, обмежену

функціональність у плані інтеграції з іншими сервісами. Це створює необхідність у розробці нових рішень, які б відповідали сучасним стандартам і потребам користувачів.

Метою роботи є покращення виконання основних операцій із проїзними картками, зокрема їх обслуговування, блокування, розблокування та продовження терміну дії пільги шляхом розробки програмної системи обліку персоналізованих проїзних карток.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- аналіз існуючих рішень для обліку проїзних карток та виявлення їхніх недоліків;
- визначення основних функціональних вимог до програмної системи;
- розробка архітектури та дизайну користувацького інтерфейсу системи;
- реалізація програмної системи з використанням сучасних технологій;
- тестування та оцінка ефективності розробленого програмного забезпечення;
- інтеграція системи з іншими сервісами міста для покращення функціональності.

Об'єктом дослідження є процеси розробки системи обліку проїзних карток.

Предметом дослідження виступає web-застосунок обліку проїзних карток, який включає в себе функціональні можливості для здійснення ремонту карток, можливість продовження пільги, продовження терміну дії, блокування та розблокування карток.

Новизна роботи полягає у наступному:

Удосконалено метод взаємодії із користувачами, який полягає у поєднанні сучасних веб-технологій та підходів до створення користувацьких інтерфейсів інтерактивних веб-застосунків для системи обліку персональних проїзних карток, що забезпечує зручну взаємодію з системою, полегшує виконання операцій з картками (таких як блокування, розблокування, ремонт, продовження пільг).

Практичне значення отриманих результатів полягає у можливості застосування розробленої системи в реальних умовах функціонування громадського транспорту. Це сприятиме покращенню обслуговування пасажирів, зменшенню витрат на адміністрування проїзних карток та оптимізації перевезень. Крім того, розроблена система може бути використана як основа для подальших досліджень у сфері транспортної автоматизації та управління громадським транспортом.

Апробація та публікація результатів бакалаврської кваліфікаційної роботи. Результати бакалаврської кваліфікаційної роботи доповідалися на конференції Електронні інформаційні ресурси: створення, використання, доступ (2022р.) і опубліковані в тезах доповіді цієї конференції, а також конференції Молодь в науці: дослідження, проблеми, перспективи (МН-2024) Правове регулювання охорони праці: обов'язки роботодавців та права працівників та Молодь в науці: дослідження, проблеми, перспективи (МН-2025) Функціональні особливості проїзних карток.

Також веб система була застосована на підприємстві, доказом цього є акт впровадження (додаток В).

1 ФОРМУЛЮВАННЯ ЗАВДАНЬ ТА АНАЛІЗ АКТУАЛЬНОСТІ РОЗРОБКИ ПРОГРАМНОЇ СИСТЕМИ ОБЛІКУ ПЕРСОНАЛЬНИХ ПРОЇЗНИХ КАРТОК

1.1 Аналіз актуальності систем обліку проїзних карток

У сучасному світі цифровізації важливою складовою стало впровадження автоматизованих систем у транспортній інфраструктурі, зокрема систем обліку персональних проїзних карток. Такі системи не лише спрощують взаємодію громадян з транспортом, а й підвищують ефективність контролю, забезпечують точну аналітику та знижують адміністративні витрати. Зважаючи на інтенсивний розвиток міст і зростання потреб у мобільності населення, роль таких рішень щороку зростає.

У Вінниці, як і в багатьох інших українських містах, спостерігається потреба в удосконаленні процесів обліку та обслуговування проїзних карток. Існуючі рішення мають обмежену гнучкість, складні для адміністрування, не підтримують персоналізовану аналітику та часто не інтегровані з іншими міськими сервісами. Через це ускладнюється обслуговування пільгових категорій, виникають труднощі з продовженням терміну дії, а також з блокуванням і розблокуванням карток.

У таких умовах доцільною є розробка веб-застосунок, що дозволяє обробляти звернення громадян щодо ремонту, блокування, продовження пільг або терміну дії карток. Саме веб-застосунок — як зручна форма клієнт-серверної системи — здатен вирішити численні проблеми, які виникають при традиційному паперовому або частково автоматизованому обліку. Він може бути доступний з будь-якої точки локальної мережі та дозволяти швидке реагування оператора, експерта чи адміністратора на кожне звернення.

Висока актуальність таких систем зумовлена також необхідністю захисту персональних даних. Системи обліку мають забезпечувати конфіденційність і безпеку інформації користувачів, а отже впровадження сучасних методів

шифрування та розмежування доступу стає критично важливим. У старих системах це питання часто ігнорується або реалізується на низькому рівні.

У рамках реформування транспортної галузі України передбачене масове впровадження електронних квитків, що робить питання обліку карток ще більш нагальним. Для того, щоб ці процеси були ефективними, необхідна централізована система обліку, яка може адаптуватися до змін нормативної бази, дозволяє вести статистику, звіти, верифікувати заявки та контролювати весь цикл обслуговування картки.

Також важливо зазначити, що автоматизовані системи обліку карток значно підвищують якість сервісу для громадян. Вони дозволяють уникати черг, мінімізують людський фактор у прийнятті рішень, знижують ризики втрати або дублювання інформації. Для людей похилого віку або осіб з інвалідністю такі рішення можуть стати вирішальними у забезпеченні зручного доступу до пільг.

Окремої уваги заслуговує зручність адміністрування. У сучасних умовах важливо мати інструмент, який дозволяє оперативно вносити зміни до бази даних, призначати ролі, створювати звіти, відслідковувати статус карток. Веб-застосунок, що працює у межах локальної мережі, є оптимальним рішенням з огляду на безпеку, простоту доступу й адміністрування.

Суттєвим є й те, що система, розроблена спеціально під інфраструктуру Вінниці, враховує її специфіку, наприклад — типи пільг, типові звернення, маршрути та категорії користувачів. Універсальні рішення, які використовуються в інших містах, не завжди враховують такі нюанси, через що виникає необхідність у кастомізації або повній заміні ПЗ.

Постійне оновлення законодавчої бази та нові вимоги до публічних сервісів потребують систем, які можна швидко змінювати або масштабувати. Саме веб-додаток дозволяє швидко інтегрувати нові функції, додавати поля до форми заявки, змінювати типи операцій або формати звітів.

Система, орієнтована на обслуговування, а не фінансові операції, має чіткий сервісний характер і фокусується на забезпеченні якісного ручного обліку. Вона не вимагає складної інтеграції з платіжними шлюзами, але водночас

повинна забезпечити точність і актуальність облікових даних, що є ключовим для адміністрування карток.

Окрім цього, автоматизований облік значно спрощує документообіг. Адміністратор або оператор може швидко знайти потрібну інформацію, роздрукувати чек, зробити звіт по місяцях, оцінити навантаження за кількістю звернень тощо. Це важливо для планування ресурсів і покращення якості послуг.

Також актуальність системи зростає у зв'язку з необхідністю боротьби з шахрайством. Можливість відстеження історії картки, блокування у разі втрати, фіксація усіх змін — усе це допомагає мінімізувати ризики несанкціонованого використання чужих карток.

Система повинна мати чітку структуру прав доступу. Учасники (оператори, експерти, адміністратори) повинні мати лише ті права, які відповідають їх функціоналу. Це дозволяє контролювати зміни, підвищити безпеку і впорядкувати роботу.

Ураховуючи, що система працює виключно у локальній мережі, вона менш вразлива до зовнішніх атак, не потребує постійного підключення до інтернету, а отже може бути ефективною навіть в умовах обмеженої інфраструктури.

Підсумовуючи, можна стверджувати, що розробка сучасного веб-застосунку для обліку персональних проїзних карток у місті Вінниці є вкрай актуальною.

1.2 Порівняльний аналіз існуючих систем

Сучасний ринок програмного забезпечення пропонує низку рішень для автоматизації сервісів ремонту, управління обліком замовлень, клієнтськими базами та технічними процесами. Розробники таких рішень намагаються задовольнити потреби як великих сервісних компаній, так і невеликих локальних майстерень. Проте аналіз показує, що навіть провідні системи мають обмеження, які не дозволяють повноцінно реалізувати специфічні функції, необхідні в окремих регіонах чи галузях.

Одним із найвідоміших рішень на ринку є система RemOnline, яка має широкий функціонал для управління сервісним обслуговуванням. Це програмне забезпечення позиціонується як повноцінна ERP-система, що включає в себе інструменти для обліку замовлень, управління персоналом, клієнтською базою, а також логістикою запчастин. Сильним боком RemOnline є хмарна архітектура: користувач може отримати доступ до системи з будь-якого пристрою, підключеного до інтернету. Крім того, інтерфейс інтуїтивно зрозумілий навіть для початківців, що пришвидшує навчання персоналу та скорочує витрати на впровадження [1].

Однак залежність RemOnline від стабільного інтернет-з’єднання створює ризики у випадку технічних збоїв або роботи в регіонах з нестабільним покриттям. Крім того, кастомізація системи обмежена. Якщо бізнес має специфічні вимоги, які виходять за межі стандартного функціоналу, впровадження змін або додавання модулів стає ускладненим або взагалі недоступним без участі технічної підтримки компанії-розробника.

Іншим прикладом комплексного рішення є система Gincore, яка створена спеціально для автоматизації процесів сервісного обслуговування. Ця система поєднує можливості CRM, ERP і WMS, що дозволяє ефективно працювати з клієнтами, здійснювати управління ресурсами підприємства, а також вести облік запчастин та матеріалів. Окремо варто відзначити, що Gincore підтримує функціонал прийому замовлень на ремонт, списання деталей, контроль етапів виконання ремонту та повідомлення клієнтів про статус замовлення [2].

Важливою перевагою Gincore є вбудована базова бухгалтерія, що дозволяє вести облік фінансових операцій без використання сторонніх систем. Також є можливість автоматизації звітності та аналітики, що є критично важливим для управлінських рішень. Проте Gincore, як і RemOnline, має досить складну архітектуру, яка може бути надмірною для невеликих підприємств. Вартість ліцензії також може стати бар'єром для впровадження у малому бізнесі.

Ще одним прикладом є АСООП (Автоматизована система обліку оплати проїзду), яка розроблена для громадського транспорту. Окремі її модулі

включають службу підтримки карток, що дозволяє здійснювати їх обслуговування, перевипуск, блокування, продовження пільг тощо. У межах АСООП використовується додаток ЛеоКарт, орієнтований на користувача. Проте функціонал для адміністративного управління є обмеженим і не дозволяє в повній мірі реалізувати сервісну підтримку в локальному середовищі. В АСООП відсутній гнучкий механізм налаштування прав доступу для різних категорій персоналу [3].

АСООП не містить модулів управління складом, повноцінної CRM-системи або інструментів з обліку ремонту пластикових карток. Уся діяльність обмежена лише обслуговуванням транспортних карт і забезпеченням доступу користувачів до особистих кабінетів. У зв'язку з цим, система не задовольняє потреби адміністратора або експерта, які мають контролювати технічний стан карт, вести аналітику, формувати звіти про виконані ремонти.

Таким чином, аналіз показує, що існуючі системи вирішують лише частину задач, пов’язаних із обслуговуванням та ремонтом карток.

Для більшої наочності проведено порівняльний аналіз основних характеристик в таблиці.

Таблиця 1.1 – Порівняльна характеристика існуючих систем

Характеристика	RemOnline	Gincore	АСООП / ЛеоКарт	CardFlow
Облік ремонтів	+	+	–	+
Підтримка роботи з клієнтами (CRM)	+	+	–	+/–
Управління запасами (WMS)	+	+	–	–
Фінансовий облік	+/–	+	–	–
Повідомлення клієнтів	+	+	+	+
Хмарна архітектура	+	+	–	– (локальна мережа)
Кастомізація	–	+/–	–	+
Складність впровадження	+/–	–	+	Низька
Вартість	Висока	Висока	Середня	Низька
Адаптація під локальні задачі	–	–	–	+

Як видно з таблиці, жодна з проаналізованих систем не забезпечує повної відповідності сучасним вимогам до автоматизованого обліку персональних проїзних карток у місті Вінниця. RemOnline і Gincore мають потужний функціонал, проте орієнтовані на інші галузі – ремонт побутової техніки або загальні сервісні центри. Вони не передбачають специфічні функції роботи з транспортними картками, як-от обробка запитів на продовження пільг, контроль термінів дії карт, або ж аудит дій з блокування/розблокування.

АСООП, з іншого боку, хоча й надає базовий набір інструментів для обслуговування карт, не виконує функції управління ремонтами, не дозволяє налаштовувати користувацькі ролі з деталізованими правами доступу, не має системи ведення обліку запчастин або реєстрації кожного етапу обслуговування картки.

Розроблена система CardFlow вирішує більшість із зазначених проблем. Вона орієнтована на конкретні локальні задачі муніципального транспорту, має простий у використанні інтерфейс, модулі для обробки заявок на ремонт, блокування, продовження пільг, а також гнучку систему ролей (оператор, експерт, адміністратор). CardFlow працює у локальній мережі, що забезпечує безпечний доступ та незалежність від інтернет-з’єднання, що є критично важливим для міських установ. Низька вартість, адаптація до локальних потреб і простота впровадження роблять цю систему конкурентною перевагою серед існуючих аналогів.

Отже, порівняльний аналіз підтверджує: існуючі програмні системи ефективні лише в рамках своєї первинної спеціалізації, і саме це обмеження відкриває перспективи для нових розробок. Розроблена система CardFlow демонструє приклад цільового, адаптивного програмного рішення, здатного забезпечити повний цикл обслуговування персональних транспортних карток у межах міста Вінниця.

1.3 Постановка завдань до роботи

Постановка завдань є важливим етапом у розробці програмного забезпечення, оскільки вона визначає основні цілі та етапи, необхідні для успішної реалізації проекту. Чітке формулювання завдань дає змогу організувати роботу команди, скоротити час розробки та забезпечити ефективне використання ресурсів.

Метою бакалаврської кваліфікаційної роботи є розробка веб-застосунку для обслуговування проїздних карток, який включає функціонал для ремонту, деблокування, продовження пільг та терміну дії карток. Система повинна забезпечити зручний і ефективний інтерфейс для користувачів та адміністраторів, а також високу безпеку даних.

Для досягнення цієї мети необхідно вирішити наступні завдання:

1. Визначити основні функціональні вимоги, які повинні бути реалізовані у веб-застосунку: Це включає функції ремонту зіпсованих карток, деблокування заблокованих карток, продовження пільг для користувачів, а також продовження терміну дії карток.

2. Розробити загальну архітектуру веб-застосунку: Створити загальну структуру системи, яка включатиме модулі для обробки запитів на ремонт, деблокування, продовження пільг і терміну дії карток, а також забезпечення ефективної взаємодії між різними компонентами системи.

3. Створити базу даних для зберігання інформації про картки: Розробити базу даних для зберігання всіх необхідних даних: інформації про картки, історії їх використання, інформації про пільги та терміни дії карток, а також даних про стан карток (активні, заблоковані тощо).

4. Реалізувати механізми безпеки для захисту даних користувачів: Забезпечити захист даних користувачів, запобігання несанкціонованому доступу та забезпечити високу надійність операцій із картками, таких як ремонт, деблокування та продовження терміну дії.

5. Розробити інтерфейс користувача для зручного управління картками: Створити інтуїтивно зрозумілий і зручний інтерфейс для користувачів

та адміністраторів, що дозволить легко виконувати операції з картками, такі як ремонт, деблокування та продовження пільг.

1.4 Порівняльний аналіз методів розв'язання поставлених завдань

Запропонована система обліку проїздних карток була реалізована за допомогою сучасних технологій, що забезпечили ефективність, масштабованість і зручність у використанні. В основі розробки лежали мови програмування Python та JavaScript, а також кілька інструментів для створення веб-застосунків. Система побудована на серверному фреймворку Flask, що забезпечує високий рівень гнучкості та ефективності для обробки запитів та роботи з базами даних.

Python є основною мовою програмування, що використовується для серверної частини. Вибір цієї мови обумовлений її популярністю серед розробників завдяки простоті, універсальності та багатій бібліотечній екосистемі, яка забезпечує швидку розробку надійних додатків. В рамках сервера, фреймворк Flask був вибраний через свою легкість і здатність легко інтегруватися з іншими інструментами та бібліотеками. Flask дає змогу швидко розробити RESTful API, що дозволяє системі ефективно взаємодіяти з іншими програмами через HTTP запити [4].

Frontend було реалізовано з використанням стандартних веб-технологій — HTML та CSS. Такий підхід дозволив створити зрозумілий, стабільний і сумісний з усіма сучасними браузерами інтерфейс. Веб-додаток має чітку структуру сторінок та зручну навігацію, що полегшує роботу користувачів із системою. За допомогою CSS були оформлені всі елементи інтерфейсу: форми, кнопки, таблиці та навігаційні панелі, що забезпечило приємний зовнішній вигляд і комфортне користування. Взаємодія з серверною частиною відбувається через HTML-форми, що дозволяє оператору, експерту або адміністратору виконувати необхідні дії без складної логіки на клієнтській стороні [5].

Для зберігання даних була обрана реляційна база даних PostgreSQL, яка відзначається високою продуктивністю, надійністю та широкими можливостями при роботі зі складними запитами. У базі даних зберігається повна інформація

про адміністраторів, операторів та експертів, типи пластикових карток, пільгові категорії користувачів, а також детальна історія операцій з кожною карткою — включаючи ремонт, блокування, деблокування, продовження терміну дії та пільг. Для взаємодії із базою даних використовується бібліотека psycopg2, яка забезпечує зручну ORM-абстракцію, дозволяє ефективно будувати запити та зменшує кількість прямого SQL-коду, спрощуючи розробку та підтримку системи.

Безпека даних є однією з основних вимог до системи. Для забезпечення високого рівня захисту використовуються методи хешування паролів і шифрування даних, що дозволяє захистити особисту інформацію користувачів від несанкціонованого доступу. Авторизація та аутентифікація користувачів здійснюється через JSON Web Token (JWT), що забезпечує безпечну роботу з користувацькими сесіями.

В процесі розробки велика увага була приділена тестуванню. Використовувались автоматизовані тести для перевірки всіх основних функцій системи, таких як створення карток, блокування, розблокування, а також функції продовження терміну дії карток. Це дозволило забезпечити стабільність роботи системи та вчасно виявляти помилки. Для тестування також використовувалися навантажувальні тести, щоб перевірити, як система працює при великому навантаженні.

Для розробки було вибрано інтегроване середовище PyCharm, яке є зручним і потужним інструментом для написання Python-коду. PyCharm має вбудовані функції для роботи з базами даних, підтримує систему контролю версій Git, автоматично доповнює код і дозволяє налагоджувати програми в реальному часі, що значно покращує продуктивність розробників.

Завдяки поєднанню таких технологій, як Python, Flask, PostgreSQL та ефективним інструментам для тестування та налагодження, вдалося створити надійну та функціональну систему обліку проїздних карток. Рішення, що були прийняті на етапі розробки, забезпечили швидкий доступ до інформації, високий рівень безпеки та зручність користування додатком для кінцевих користувачів.

1.5 Висновок

У розділі розглядається концепція та значення автоматизації обліку та управління персональними проїзними картками у місті Вінниця. Система обліку карток є важливим елементом для забезпечення ефективного та прозорого управління транспортними послугами. Впровадження сучасних технологій дозволяє значно спростити процеси обслуговування карток, зменшити людський фактор та підвищити рівень надійності.

Основним завданням системи є забезпечення зручності для користувачів при обслуговуванні карток та зменшення часу, необхідного для виконання операцій. Це включає в себе не лише ремонт карток, але й їх блокування, деблокування, продовження пільг та терміну дії. Враховуючи специфіку потреби у такому сервісі, він має бути доступним для використання лише кваліфікованими особами: адміністратором, оператором та експертом.

Процес автоматизації вимагає розробки ефективної системи управління даними, що повинна враховувати специфіку роботи з пластиком, можливість дистанційного доступу до певних функцій, а також оперативне оновлення бази даних. Водночас система повинна бути гнучкою, забезпечуючи можливість налаштування під індивідуальні потреби та зміни в нормативно-правовому середовищі.

Один з ключових аспектів — це безпека інформації. Дані користувачів повинні бути захищені від несанкціонованого доступу, що вимагає застосування сучасних технологій шифрування та аутентифікації. Крім того, система повинна бути стійкою до технічних збоїв і дозволяти оперативно відновлювати дані в разі виникнення неполадок.

Аналіз існуючих систем управління обліком карток показує, що більшість рішень орієнтовані на автоматизацію окремих етапів, таких як блокування або ремонт карток. Однак, відсутність комплексних рішень, які об'єднують усі етапи в одну систему, є значною проблемою для ефективності управління. Це створює необхідність у розробці інтегрованої системи, здатної забезпечити повний цикл обслуговування карток з мінімальними витратами часу та ресурсів.

Зокрема, необхідно враховувати можливість інтеграції з існуючими міськими системами, такими як електронні платіжні системи або системи контролю доступу до транспорту. Це дозволить забезпечити зручність для користувачів, адже вони зможуть використовувати одну картку для кількох послуг.

Впровадження такої системи потребує ретельного аналізу існуючих рішень, вивчення переваг та недоліків конкурентних технологій, а також розробки детального плану інтеграції. Враховуючи швидкий розвиток інформаційних технологій та постійні зміни в нормативно-правовій сфері, важливо передбачити можливість масштабування системи на майбутнє.

Таким чином, розробка та впровадження автоматизованої системи обліку персональних проїзних карток для міста Вінниця є необхідним кроком для покращення обслуговування громадян та підвищення ефективності роботи транспортної системи. Це дозволить не лише оптимізувати робочі процеси, але й забезпечити високий рівень безпеки даних та зручності для користувачів.

2 АНАЛІЗ СТРУКТУРИ АЛГОРИТМІВ ТА ФОРМУВАННЯ ПРОБЛЕМАТИКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз проблематики обліку проїзних карток

Для успішної розробки програмної системи обліку персональних проїзних карток м. Вінниця необхідно ретельно вивчити та проаналізувати наявні проблеми в цій сфері. Важливо зрозуміти основні виклики, з якими стикаються користувачі, контролюючі органи та транспортні компанії, а також виявити шляхи їх вирішення. Це включає всебічне дослідження технічних, функціональних та організаційних аспектів [6].

Перше, що потрібно ретельно проаналізувати, це функціональні труднощі, з якими можуть стикатися користувачі. Наприклад, проблеми з верифікацією картки, її втрата або пошкодження, труднощі з блокуванням та розблокуванням картки чи продовження терміну дії можуть суттєво вплинути на ефективність використання карткової системи.

Система обліку проїзних карток є важливим інструментом для управління картками, що використовуються для проїзду в громадському транспорті. У нашому випадку, система орієнтована на облік та обслуговування карток, що включає операції з їх ремонту, блокування, розблокування, продовження пільг і терміну дії. Для успішної розробки та впровадження такої системи важливо детально проаналізувати існуючі проблеми та виявити шляхи їх вирішення.

Основною проблемою є забезпечення надійної роботи механізмів блокування та розблокування карток. Необхідно гарантувати, що картки можуть бути ефективно заблоковані у випадках втрати або крадіжки, а також швидко розблоковані, якщо власник картки відновить її. Важливим аспектом є забезпечення безпеки цих операцій, щоб уникнути несанкціонованого доступу до карток або маніпуляцій з їхнім станом.

Ще однією важливою проблемою є ефективне продовження терміну дії карток та пільг. Часто система може не оновлювати статус картки вчасно, що може призвести до того, що користувач не зможе скористатися пільговим

тарифом або відмовлятимуть у поїздці через невірну інформацію про термін дії картки. Тому необхідно забезпечити автоматичне та точне оновлення терміну дії картки, а також пільг, щоб користувачі могли без проблем користуватися своїми картками.

Не менш важливим є механізм ремонту карток. У разі пошкодження картки важливо, щоб система дозволяла швидко обробляти заявки на її відновлення та заміну. Це дозволить користувачам мінімізувати незручності, пов’язані з поломкою картки, та зберегти свої пільги та термін дії картки без втрат.

З технічної точки зору, система повинна бути стабільною та швидко реагувати на запити клієнтів. Кожна операція, така як блокування, розблокування, ремонт картки або продовження пільги, повинна виконуватись оперативно, без затримок, щоб користувачі не стикались із затримками або технічними збоєм. Регулярне оновлення системи та перевірка її на вразливості також є важливими аспектами для забезпечення безперебійної роботи.

Інтерфейс системи повинен бути простим і зрозумілим для користувачів, незалежно від їх технічної підготовки. Потрібно створити інтуїтивно зрозуміле меню для виконання операцій з картками, таких як блокування, розблокування, продовження терміну дії, а також для отримання статусу картки. Це допоможе знизити кількість помилок і покращить взаємодію користувачів з системою.

Усі ці аспекти потребують ретельного аналізу і дослідження, щоб забезпечити розробку ефективної та надійної системи обліку проїзних карток, яка буде працювати стабільно, безпечно та зручно для користувачів. Розуміння цих проблем та їх вирішення дозволить створити конкурентоспроможну систему, яка задовольнятиме всі вимоги як користувачів, так і операторів транспортних послуг.

Підсумовуючи, всебічне дослідження проблематики є критично важливим для успішної реалізації проєкту. Ретельний аналіз дозволить визначити основні виклики та знайти ефективні шляхи їх вирішення, що в свою чергу сприятиме створенню високоякісної та конкурентоспроможної системи обліку проїзних карток у м. Вінниця.

2.2 Дослідження функціональних можливостей системи

Розробка програмної системи обліку персональних проїзних карток міста Вінниці передбачає створення зручного інструменту для ведення обліку, організації обслуговування та ремонту таких карток. Основною метою є підтримка обробки звернень громадян, які потребують ремонту чи оновлення інформації про свою транспортну картку. Вона створена для покращення координації між користувачем, експертом та адміністратором, зменшення навантаження на операторів та зручного контролю процесів обслуговування.

Функціональність системи зосереджена на чотирьох основних напрямках: подання заявок на ремонт картки, обробка звернень щодо блокування/розблокування, продовження пільгового статусу, а також ведення обліку технічного стану карток.

Кожна заявка у системі супроводжується атрибутами — тип звернення, ПІБ користувача, ідентифікатор картки, статус обробки, а також дата надходження й коментар оператора або експерта. Це дозволяє здійснювати базовий контроль над етапами розгляду звернення та вчасно реагувати на запити. Система не приймає рішень самостійно, а лише надає зручне середовище для ручної обробки запитів.

Особлива увага приділяється модулям ремонту та повторного випуску картки. При подачі відповідного звернення користувач вказує характер поломки або проблему, після чого заявка проходить попередню перевірку адміністратором. Надалі вона може бути передана експерту для технічного аналізу. Рішення щодо ремонту приймається вручну, а інформація про стан картки оновлюється в загальній таблиці системи.

Також реалізовано модуль блокування та розблокування карток, який дає змогу адміністратору вручну змінювати статус картки на підставі заявки користувача. У випадках втрати або підозри у шахрайстві картку можна тимчасово заблокувати, а після підтвердження особи — відновити доступ. Уся інформація про подібні дії фіксується для подальшого аналізу та архівації.

Ще однією важливою функцією є продовження терміну дії пільг. Система дозволяє адміністратору переглянути дані про тип пільги користувача та оновити її термін відповідно до документів, поданих через офлайн або електронний канал. Ця функція виконується вручну, проте вся відповідна інформація фіксується в системі, що дозволяє уникнути помилок і дублікатів.

Інтерфейс системи структуровано таким чином, щоб забезпечити швидкий доступ до основних функцій обробки заявок. Елементи інтерфейсу згруповані за функціональністю: подання заявок, перегляд статусів, керування картками. Кожен статус заявки відображається у вигляді текстового маркера, що дозволяє адміністратору або експерту оперативно визначити поточний етап її обробки. Передбачена таблиця зі списком звернень, яка підтримує сортування та фільтрацію за ключовими параметрами (дата, тип, тощо).

З боку адміністратора передбачено можливість створення, редагування та архівації записів про картки. Це дає змогу підтримувати базу даних в актуальному стані та, за необхідності, здійснювати перевірку історії звернень. Водночас реалізована можливість експорту таблиць у форматі Excel для подальшої звітності.

Система також підтримує сортування та фільтрацію заявок за різними параметрами: тип операції, статус, дата, ПІБ користувача тощо. Це дає змогу швидко знаходити потрібну інформацію, що особливо корисно в умовах великого потоку звернень. Така функціональність є важливою складовою ефективної роботи експерта або адміністратора.

Щодо безпеки, система реалізує базові заходи контролю доступу. Для захисту персональних даних передбачено розмежування прав доступу: адміністратори мають повний доступ до заявок і налаштувань, експерти — лише до технічної частини обробки, а клієнти — тільки до подачі заявок та перегляду їхнього статусу.

Система працює виключно в локальній мережі, що забезпечує підвищений рівень безпеки та контроль над даними. Такий підхід унеможливлює

несанкціонований доступ ззовні, мінімізує ризики витоку персональної інформації та дозволяє обслуговувати звернення лише користувачам.

Функціональність також охоплює журнал змін — список усіх змін, внесених до конкретної заявки або картки. Це дозволяє проводити аудит дій операторів і забезпечує прозорість процесів обслуговування. У разі помилкової дії журнал змін допомагає відновити правильний стан запису.

Окремим блоком передбачено систему сповіщень: користувач отримує повідомлення про зміну статусу заявки, а експерт — про надходження нових звернень. Це дає змогу оперативно реагувати на нові запити та зменшити час очікування для клієнта.

Отже, функціональні можливості системи орієнтовані на полегшення ручного процесу обліку та обслуговування проїзних карток. Система але значно спрощує документообіг і комунікацію між сторонами. Такий підхід дає змогу організувати облік у зручному цифровому форматі, що суттєво підвищує ефективність у повсякденній роботі сервісних служб.

2.3 Аналіз та розробка інтерактивних можливостей програмної системи

Інтерактивні можливості програмної системи розроблені з урахуванням функціональних обов’язків основних акторів — клієнта, оператора, експерта та адміністратора. Система є інструментом ручного супроводу процесу ремонту карток і забезпечує зручну взаємодію між усіма учасниками.

Інтерфейс системи побудований відповідно до ролей. Кожен актор бачить лише ті елементи, які йому необхідні для виконання своїх функцій. Наприклад, оператор має доступ до реєстрації заявок, перегляду карток, передачі їх на ремонт. Експерт бачить лише картки, передані на обробку, та може змінити їх статус після виконання ремонту. Адміністратор має доступ до налаштувань, а клієнт взагалі не працює безпосередньо з системою — звернення здійснюється через оператора.

Подача заявки починається з клієнта, який передає картку оператору. Оператор виконує початкову обробку: вносить дані до системи через інтерактивну форму, де зазначає тип звернення, серійний номер картки та короткий опис проблеми. Після заповнення всіх обов’язкових полів система автоматично генерує унікальний номер заявки.

Заявка зберігається у загальному журналі, до якого мають доступ як оператор, так і експерт. Всі заявки мають фіксований статус, який змінюється залежно від дій виконавців. Статуси включають: «Прийнято», «В ремонті», «Відремонтовано », «Видана». Ці значення оновлюються вручну відповідними акторами через інтерфейс системи.

Для зручності обробки звернень система надає таблицю з усіма заявками. Передбачено можливість фільтрації та сортування за статусом, датою, номером картки або оператором. Це дає змогу швидко знайти потрібну заявку або контролювати хід виконання робіт у межах змін.

Експерт після отримання картки має можливість перейти до відповідного звернення, ознайомитися з описом проблеми та після виконання ремонту оновити статус заявки. Оновлення статусу супроводжується внесенням дати завершення та, за необхідності, додаткового коментаря. Це дозволяє фіксувати результати робіт у системі без зайвої документації.

Після ремонту оператор отримує оновлену інформацію в таблиці заявок і може видати картку клієнту. У системі передбачено позначення, що картка готова до видачі. Після фактичної передачі картки оператор змінює статус на «Видана», що закриває звернення.

Адміністратор має окремий доступ до модуля налаштувань, де керує створенням користувачів, призначенням ролей, скиданням паролів та моніторингом активності в системі. Він не втручається у процес ремонту, але забезпечує безперебійну роботу платформи.

Для відстеження дій передбачено журнал подій, який автоматично реєструє всі зміни у статусах заявок та дії користувачів із зазначенням дати, часу

та ролі, яка здійснила зміну. Це дозволяє адміністратору контролювати якість виконання процедур і при необхідності відновити хронологію обробки.

Інтерактивність системи також полягає в можливості оновлення таблиць у реальному часі. Після кожної дії оператор або експерт бачить актуальний стан заявки без необхідності оновлення сторінки або повторного входу в систему. Це пришвидшує роботу й дозволяє уникнути дублювання.

Додатково, у системі реалізовані попередження про відсутність обов'язкових даних при створенні заявки. Це знижує ризик втрати важливої інформації під час первинного введення. Всі поля, обов'язкові до заповнення, відмічені відповідними візуальними підказками.

У випадку технічних проблем, адміністратор має змогу вручну редагувати або деактивувати заявку. Також доступна функція ручного закриття старих звернень, що дозволяє підтримувати актуальність робочої таблиці без втрати історичних даних.

Таким чином, функціональні можливості системи чітко узгоджені з ролями та завданнями кожного учасника процесу обслуговування карток. Вся логіка взаємодії побудована на ручних діях з елементами автозаповнення, що полегшує роботу операторів та експертів. Інтерфейс розроблений таким чином, щоб скоротити кількість кроків при обробці звернень і забезпечити швидкий доступ до ключових функцій.

2.4 Розробка загальної моделі програмного забезпечення

Загальна модель програмного забезпечення побудована за принципами інтуїтивної навігації, модульності та автоматизації обробки запитів. Система є веборієнтованою, що дозволяє працювати з нею через браузер, без потреби встановлення окремих додатків. Інтерфейс адаптований для операторів, які взаємодіють із клієнтами в процесі прийому, обробки та видачі проїзних карток.

Основна структура вебдодатку поділена на два рівні: верхнє горизонтальне меню (навігаційна панель) та основна зона контенту. У верхній панелі розміщено такі розділи:

- Адмін панель — доступна лише адміністраторам для налаштування прав доступу та управління обліковими записами.
- Заявки — основна таблиця всіх зареєстрованих запитів, де видно статуси та дані клієнтів.
- Блок / Деблок — функціонал для блокування або відновлення карток.
- Ремонт — доступ до заявок, які знаходяться на етапі технічного обслуговування.
- Блок лист — окремий модуль для відстеження заблокованих карток.
- Таблиця — загальна форма для швидкого доступу до бази карток.

У правій частині верхньої панелі також розташовано випадаючий список вибору принтера (наприклад, «2 Стіл») і відображення авторизованого користувача (у прикладі — «Андрій Томай»), з можливістю вийти із системи.

На головній сторінці відображається заголовок «Створити заявку» та форма для введення даних. Усі поля форми структуровані та зрозумілі для оператора:

- Номер картки — вводиться вручну або зчитується зі сканера.
- Статус OTPUZ — Статус який має картка в системі АСООП.
- Ім’я, Прізвище, Телефон — персональні дані власника картки.
- Скарга — випадаючий список із типом несправності (за необхідності).
- Примітки — додаткові уточнення, які не передбачені іншими полями.

У нижній частині форми є дві кнопки: «Подати заявку» — зберігає дані у базі та запускає друк, і «Знайти картку» — для пошуку вже зареєстрованих карток у системі.

На рисунку 2.1 представлена UML-діаграма, що демонструє взаємодію основних компонентів системи.

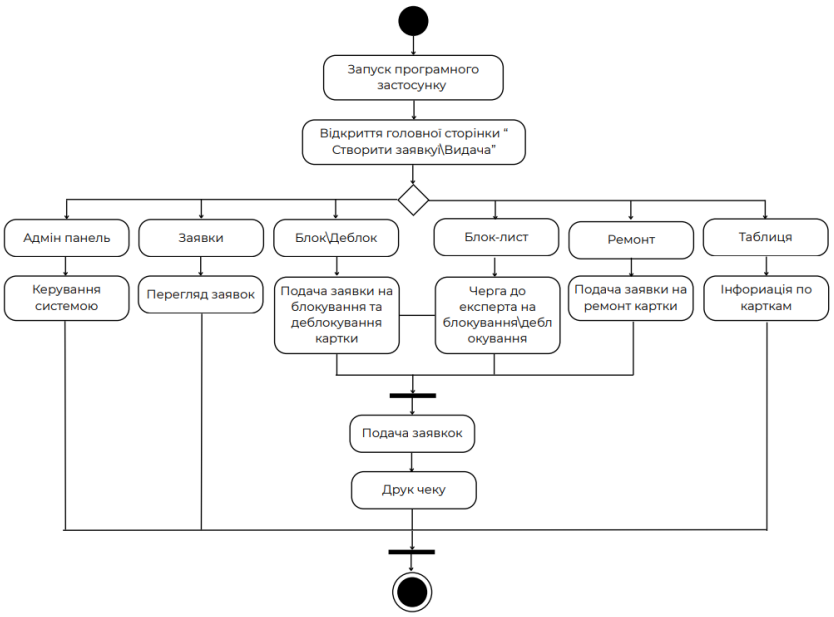

Рисунок 2.1 – UML-діаграма активності застосунку

Після успішного подання заявки система автоматично генерує та друкує чек, який слугує підтвердженням для клієнта. Структура чеку містить:

- штрих-код для швидкої ідентифікації замовлення;
- назву сервісу: Муніципальна картка вінничанина;
- адресу та контактний номер офісу: Соборна 36, (0800)330-155;
- прізвище та ім’я клієнта;
- номер картки;
- унікальний номер замовлення, що формується системою (наприклад: 50657);
- дату та час прийому (формат: 2025-03-25 16:06);
- скаргу або вказівку не вказано, якщо поле не заповнено.

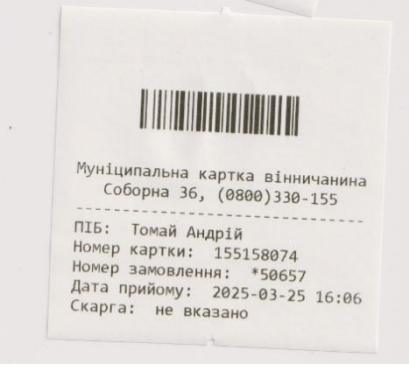

Рисунок 2.2 – Фото чеку

Цей чек друкується автоматично через обраний принтер, що прискорює обслуговування клієнтів і зменшує ризики втрати інформації. У майбутньому такий підхід також дозволяє реалізувати інтеграцію з обліковими модулями або CRM-системами.

Отже, модель програмного забезпечення охоплює повний цикл обслуговування клієнта — від подання заявки до її фізичного підтвердження, з можливістю відстеження всіх етапів у системі. Це забезпечує прозорість, контроль та ефективність у роботі з персоналізованими картками.

2.5 Дослідження алгоритмів роботи системи

У процесі розробки веборієнтованої системи обліку персональних проїзних карток важливе значення має дослідження алгоритмів її роботи. Це дає змогу забезпечити стабільну роботу основних модулів, оптимізувати обробку запитів, підвищити зручність використання системи та уникнути помилок на етапі її експлуатації. Алгоритмічна база проєкту охоплює три ключові блоки: прийом карток, видача карток, а також блокування/розблокування.

Перший критичний модуль — алгоритм прийому карток. Цей блок дозволяє оператору створити нову заявку, що містить такі дані: номер картки, статус OTPUZ (тип звернення), ім’я, прізвище, телефон, скаргу (якщо є) та примітки. На рисунку 2.3 представлено блок-схему, яка демонструє етапи обробки заявки. Після заповнення всіх полів відбувається валідація даних: перевірка наявності номера картки в системі, коректність формату телефону, обов’язковість заповнення ключових полів. Якщо перевірку пройдено успішно — система зберігає заявку в базі даних і запускає процес друку чеку.

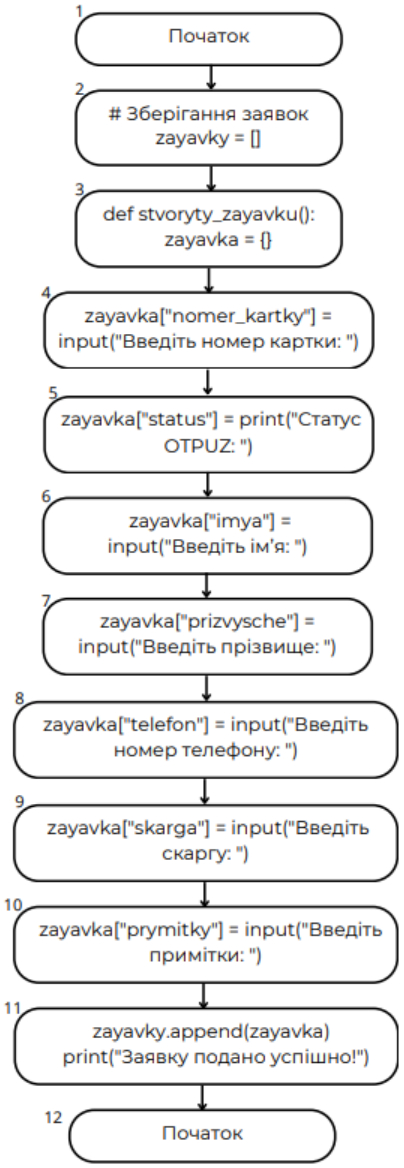

Рисунок 2.3 – Блок-схема алгоритму прийому карток

Другий важливий модуль — алгоритм видачі карток, що забезпечує завершення циклу обслуговування клієнта. Робота модуля починається з пошуку заявки за номером картки, номером телефону або унікальним номером замовлення. Після цього система виводить статус заявки, дані клієнта, дату подання та, якщо є, дату завершення ремонту. За потреби можна повторно надрукувати чек. Якщо заявка готова до видачі, система змінює її статус і оновлює відповідні записи. Алгоритм гарантує точність видачі та дозволяє відстежувати історію обробки картки. Візуалізація цього процесу представлена на рисунку 2.4.

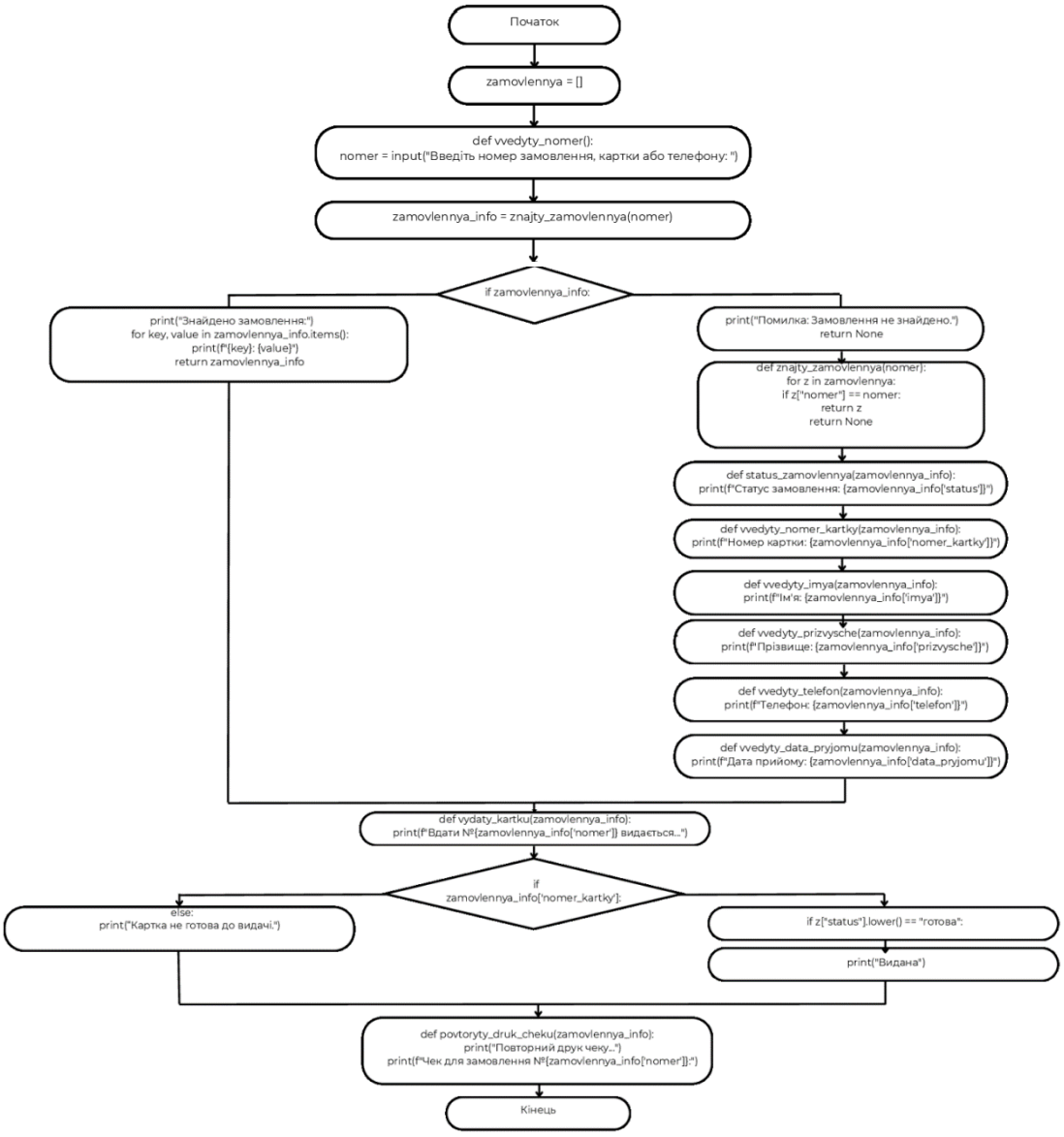

Рисунок 2.4 – Блок-схема алгоритму видачі карток

Алгоритмічна структура системи дозволяє забезпечити прозору та логічну взаємодію між модулями, а також спростити роботу операторів за рахунок автоматизації рутинних процесів. Застосування блок-схем у процесі розробки сприяло чіткому визначенню послідовності дій у кожному сценарії, що значно полегшило тестування та виявлення потенційних помилок. Такий підхід до організації логіки роботи забезпечує масштабованість та легку адаптацію системи до змін у майбутньому.

2.6 Висновки

У розділі 2 розглянуто аналіз структури алгоритмів та формування функціональних рішень, що лежать в основі розробки веб-застосунку для обліку персональних проїзних карток у місті Вінниця. У процесі дослідження було детально розглянуто як архітектурні, так і логічні аспекти програмної системи, що дозволило обґрунтувати її технічну доцільність, відповідність поставленим вимогам та перспективність у практичному застосуванні.

Основною особливістю створеної системи є оптимізована багаторівнева структура, яка забезпечує чітке розмежування функціоналу між адміністративною частиною, ролями операторів та експертів. Це дозволяє ефективно організувати роботу всередині сервісного центру, що обслуговує картки, та мінімізувати ймовірність дублювання дій або помилок при обробці звернень громадян. Окремо було розроблено модулі для кожного типу операцій — продовження терміну дії картки, відновлення після пошкодження, блокування, деблокування тощо.

На основі аналізу потреб користувачів системи (виключно працівників сервісного центру) було сформовано логічну модель, яка передбачає збереження, обробку та оновлення даних у режимі реального часу. Обрана технологічна база — Python із використанням фреймворку Flask у поєднанні з HTML та CSS — забезпечує гнучкість реалізації, зручність у масштабуванні, а також зменшує витрати на подальше обслуговування та супровід.

У рамках розділу було детально проаналізовано алгоритми обробки заявок на ремонт або обслуговування карток. Зокрема, розроблено механізм верифікації інформації за унікальним ID картки, що дозволяє автоматично ідентифікувати поточний статус, історію звернень і відповідну категорію пільговості. Це значно пришвидшує процес прийняття рішень експертом та підвищує загальну продуктивність обслуговування.

Також проаналізовано алгоритм логування дій користувачів системи. Це забезпечує прозорість та контроль процесів, дозволяє виявляти аномалії у функціонуванні програмного забезпечення, а також формує підґрунтя для

внутрішніх аудитів та удосконалення процедур сервісу. Всі зміни у статусі картки супроводжуються записом у лог, що гарантує збереження важливої інформації у разі необхідності зворотного аналізу.

Важливе значення має модуль інтеграції із внутрішньою базою даних, який реалізовано з урахуванням сучасних вимог до безпеки. Впроваджено перевірку автентичності сесії кожного користувача та обмежено доступ до операційної частини системи відповідно до ролі. Таким чином, досягнуто високого рівня інформаційного захисту, що є критично важливим при роботі з персональними даними.

Результати проведеної роботи демонструють відповідність запропонованого технічного рішення практичним потребам муніципального сервісного центру. Враховано специфіку процесів обробки пільгових категорій населення, актуальні вимоги до збереження облікових записів та потребу в гнучкому адмініструванні. Завдяки модульній структурі система може бути доповнена новими функціями без порушення цілісності програмного коду.

Особливу увагу приділено створенню зручного інтерфейсу для внутрішнього користувача. Кожен модуль має інтуїтивно зрозумілий дизайн, що сприяє швидкому навчанню персоналу та зменшенню кількості помилок при взаємодії з системою. Функціонал адаптовано під специфіку завдань операторів та експертів, що забезпечує чітку логіку дій на кожному етапі обробки заявки.

У межах цього розділу було також запропоновано варіанти оптимізації певних процедур через впровадження автоматичних перевірок, фільтрів та сортування, що дозволяє зменшити навантаження на працівників сервісного центру та підвищити швидкість обслуговування. Усі ці зміни спрямовані на вдосконалення якості сервісу та задоволення вимог муніципальної інфраструктури.

Таким чином, проведений аналіз підтвердив раціональність розробленої структури веб-додатку та довів доцільність застосування запропонованого алгоритмічного підходу в умовах сучасного українського міста. Система володіє достатнім потенціалом для подальшої адаптації під інші міста або регіони, що

свідчить про її універсальність та перспективність як інструменту управління транспортними сервісами.

Загалом, результати роботи у другому розділі дозволили не лише сформувати чітке бачення архітектури системи, а й визначити подальші напрями її розвитку. Створений фундамент є надійною основою для впровадження цифрових сервісів у сфері муніципального транспорту та автоматизації процедур обслуговування пільгових категорій громадян.

3. РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ПРОГРАМНОЇ СИСТЕМИ ОБЛІКУ ПЕРСОНАЛЬНИХ ПРОЇЗНИХ КАРТОК

3.1 Обґрунтування вибору засобів реалізації системи

Для розробки програмної системи, яка забезпечує автоматизацію обліку персональних проїзних карток, було прийнято рішення використовувати мову програмування Python у поєднанні з фреймворком Flask для серверної частини, а також HTML і CSS для розробки інтерфейсу користувача. Для роботи з проектом була обрана інтегрована середа розробки (IDE) PyCharm, що забезпечує зручні інструменти для програмування на Python і підтримує ефективну роботу з іншими технологіями. Нижче наведено порівняння обраних засобів із іншими розглянутими варіантами [7, с.144-148].

Python — це одна з найбільш популярних мов програмування, відома своєю простотою, високою читабельністю та універсальністю. Вибір Python був обумовлений його широким застосуванням у розробці веб-додатків, обробці даних, а також великою кількістю наявних бібліотек, що дозволяють швидко вирішувати завдання будь-якої складності. Python надає можливість швидко розробляти як серверну частину, так і інтегрувати зовнішні сервіси для обробки даних, що особливо важливо для розробки системи для обліку карток. Однією з ключових переваг є велика кількість готових бібліотек і фреймворків, що дозволяють ефективно працювати з базами даних, реалізовувати API та інші сервіси. Також Python добре підтримує роботу з бібліотеками для роботи з веб-технологіями, що спрощує процес створення веб-застосунків.

Flask, вибраний як фреймворк для серверної частини, є легким та гнучким рішенням для розробки веб-додатків. Його популярність серед розробників Python зумовлена можливістю створювати потужні та масштабовані веб-системи без зайвих налаштувань, що є великою перевагою для проекту. Flask дозволяє швидко реалізувати необхідний функціонал і налаштувати систему відповідно до конкретних вимог. Завдяки своїй легкості та мінімалістичному підходу, Flask дає можливість зосередитися на розробці функціональних можливостей застосунку,

без потреби конфігурувати складніші частини, які є в інших фреймворках [8, с.60-62].

HTML і CSS були обрані для створення інтерфейсу користувача, оскільки вони є стандартом для розробки веб-сторінок. HTML дозволяє структурувати контент і забезпечувати логічну організацію елементів на сторінці, в той час як CSS забезпечує гнучкість в оформленні і адаптивному дизайні веб-сторінок. Ці технології дають можливість створити зручний, інтуїтивно зрозумілий інтерфейс, що є важливою складовою частиною будь-якої сучасної веб-системи. За допомогою HTML і CSS можна створити такі елементи інтерфейсу, як форми для введення даних, відображення списків і таблиць, що є необхідними для роботи з персональними проїзними картками.

Для розробки і підтримки системи було обрано PyCharm як інтегровану середу розробки (IDE), оскільки цей інструмент є одним з найбільш потужних і зручних для програмування на Python. PyCharm має багато корисних функцій, таких як підсвічування синтаксису, автодоповнення, інтеграція з системами контролю версій, що значно підвищує ефективність роботи розробника. Ця IDE забезпечує зручне тестування коду і підтримує інтеграцію з іншими технологіями, що використовуються в проекті, такими як HTML, CSS і Flask.

Таблиця 3.1 – Порівняння засобів реалізації програмного застосунку

Характеристики	Python + Flask	JavaScript (Node.js + Express)	PHP (Laravel)	Ruby (Ruby on Rails)
Простота використання	+	–	–	–
Гнучкість фреймворка	+	+	–	+
Ширина підтримки бібліотек	+	+	+	–
Швидкість розробки	+	–	+	–
Сумісність з іншими технологіями	+	+	+	+
Підтримка масштабованих додатків	+	+	+	+
Легкість у підтримці	+	–	–	–
Підтримка адаптивного дизайну	+	+	+	+

З огляду на переваги Python і Flask, вони були вибрані як основні технології для розробки програмного застосунку для обліку проїзних карток. Python, завдяки своїй простоті, швидкості розробки і широким можливостям для інтеграції з різними бібліотеками та фреймворками, став ідеальним вибором для реалізації серверної частини. Flask, зі своєю легкістю та гнучкістю, дозволив зберегти мінімум налаштувань і забезпечити необхідну функціональність для проекту. HTML і CSS забезпечили створення зручного і адаптивного інтерфейсу для користувача, а PyCharm, як IDE, значно полегшив процес розробки завдяки своїм потужним функціям і зручності використання.

Сучасні можливості мов програмування значно впливають на здатність системи ефективно виконувати завдання, пов'язані з автоматизацією обліку та обробкою великих обсягів даних. Python, як одна з найпопулярніших мов програмування, надає численні переваги для реалізації подібних проектів завдяки своїй простоті, універсальності та підтримці великої кількості бібліотек і фреймворків. Завдяки Python можна швидко створювати функціональні прототипи і масштабовані системи, що особливо важливо для проектів, які потребують обробки та аналізу великих обсягів даних.

Особливе значення має використання Flask для серверної частини додатку. Flask дозволяє реалізувати серверну логіку з мінімальними налаштуваннями, що дає змогу зосередитись безпосередньо на створенні функціональних можливостей і швидко інтегрувати потрібні сервіси. Цей фреймворк також ідеально підходить для створення веб-сервісів, що дозволяє реалізувати необхідний API для взаємодії з базами даних і зовнішніми модулями. Завдяки простоті Flask стає ідеальним вибором для невеликих та середніх проектів, де важлива гнучкість і можливість швидко вносити зміни у логіку додатка [9, с.114-120].

Вибір HTML і CSS для розробки інтерфейсу користувача був зумовлений їх універсальністю та наявністю широкої підтримки сучасних стандартів веб-дизайну. HTML дає змогу створювати чітку структуру сторінок, тоді як CSS дозволяє додати стилі та зробити інтерфейс привабливим і зручним для

користувача. Взаємодія цих двох технологій дає змогу створити адаптивний інтерфейс, що зручно працює на різних пристроях, що є важливим аспектом при розробці веб-додатків.

PyCharm, як інтегрована середа розробки (IDE), була обрана завдяки своїм потужним інструментам для Python, а також підтримці багатьох інших технологій, що використовуються в проекті. PyCharm дозволяє ефективно писати, тестувати та налагоджувати код, а також зручно працювати з бібліотеками і фреймворками, що використовується для роботи з Flask, HTML і CSS. Інструменти для аналізу коду та інтеграція з системами контролю версій (наприклад, Git) значно полегшують командну розробку і підтримку проекту на кожному етапі його реалізації.

Загалом, поєднання Python, Flask, HTML, CSS та PyCharm дозволяє забезпечити високу ефективність, гнучкість і швидкість розробки. Це дозволяє зосередитись на вирішенні важливих бізнес-завдань, а не на вирішенні технічних проблем з реалізацією інфраструктури чи підтримкою системи. Завдяки Python розробка може бути швидкою і безпечною, а використання Flask дозволяє швидко впроваджувати необхідний функціонал. HTML і CSS забезпечують привабливий і адаптивний інтерфейс, що позитивно позначається на користувацькому досвіді. PyCharm забезпечує ефективний процес розробки, що зменшує час на налагодження та тестування, що в свою чергу сприяє прискоренню розробки та зниженню витрат.

Таке поєднання технологій є оптимальним для створення веб-додатка, який буде не лише ефективним і масштабованим, але й простим у підтримці та вдосконаленні в майбутньому.

3.2 Обґрунтування вибору середовища розробки веб-застосунку

Розробка веб-застосунку для автоматизованого обліку персональних проїзних карток у місті Вінниця потребує обґрунтованого вибору середовища розробки, що відповідає вимогам проєкту, обраним технологіям та забезпечує ефективність на всіх етапах створення системи. Основним завданням було

обрати середовище, яке б максимально підтримувало мови програмування Python, HTML, CSS та фреймворк Flask, а також забезпечувало зручність розробки, тестування, відлагодження і подальшої підтримки програмного коду.

У процесі аналізу були розглянуті такі популярні середовища розробки:

1. PyCharm — це інтегроване середовище розробки, орієнтоване на мову програмування Python. Воно підтримує фреймворки Flask та Django, має зручне автодоповнення, вбудовані інструменти для налагодження, тестування, роботу з базами даних, шаблонами Jinja2 та Git. PyCharm дозволяє легко організувати структуру веб-проєкту, а також містить візуальні підказки, що значно прискорює процес розробки. Середовище пропонує як безкоштовну Community-версію, так і розширену Professional-версію з підтримкою веб-розробки [10].

2. Visual Studio Code — це легке, гнучке та розширюване середовище з широкою підтримкою різних мов програмування через систему розширень. За допомогою відповідних плагінів, таких як Python, Flask Snippets, Jinja та ін., можна налаштувати комфортне середовище для розробки на Flask. Проте, порівняно з PyCharm, VS Code не має глибокої інтеграції з Python-фреймворками і вимагає більшої кількості налаштувань вручну [11].

3. Sublime Text — швидкий текстовий редактор, який підтримує плагіни для Python і HTML/CSS. Попри свою простоту, він не надає повного спектра інструментів для професійної розробки веб-застосунків і вимагає додаткових розширень, які лише частково компенсують відсутність повноцінної інтеграції з фреймворками [12].

4. Atom — ще один редактор з відкритим вихідним кодом, який підтримує розширення для Python та Flask. Хоча Atom є достатньо зручним для написання коду, він поступається в продуктивності та функціональності іншим середовищам, зокрема при роботі зі складними проєктами [13].

Усі середовища розробки були проаналізовані за критеріями підтримки Python/Flask, зручності розробки, розширюваності, інструментів для тестування і відлагодження, а також наявності документації та активності спільноти.

Результати аналізу наведено у таблиці 3.2.

Таблиця 3.2 – Порівняльна таблиця аналогів середовищ розробки

Характеристика	PyCharm	Visual Studio Code	Sublime Text	Atom
Підтримка Python/Flask	+	+	–	–
Підтримка HTML/CSS	+	+	+	+
Автодоповнення та підказки	+	+/–	–	+/–
Інструменти для тестування та налагодження	+	+/–	–	–
Інтеграція з Git	+	+	+	+
Підтримка Jinja2	+	–	–	–
Зручність інтерфейсу	+	+	+	+
Активна спільнота	+	+	+	+

Після порівняння можна зробити висновок, що найбільш оптимальним вибором для реалізації веб-застосунку "CardFlow" є середовище PyCharm. Воно повністю відповідає вимогам до створення серверного веб-додатку з використанням Python та Flask, забезпечує стабільність, зручність та підтримку всіх необхідних інструментів розробника. Саме це середовище дозволяє ефективно реалізувати всі функції системи, мінімізуючи витрати часу на налаштування і переключення між різними інструментами.

3.3 Розробка модуля ролі системи

Модуль ролей у системі управління персональними проїзними картками для міста Вінниця реалізовано з використанням бази даних PostgreSQL для ефективного збереження та управління інформацією про користувачів і їх ролі в системі. Цей модуль має на меті забезпечити належний рівень доступу до функціоналу застосунку для різних категорій користувачів, таких як адміністратори, оператори та експерти.

Модуль реалізує механізм авторизації і автентифікації користувачів, а також перевірку їхніх прав на виконання тих чи інших операцій. Усі операції, пов'язані з ролями, здійснюються на основі таблиць у PostgreSQL, що зберігають інформацію про користувачів та їх права. Розроблена система дозволяє чітко

визначити, які дії доступні кожній ролі, і контролювати доступ до важливих функцій.

У межах модуля була створена таблиця roles, що містить основні ролі системи, а також таблиця user_roles, яка здійснює зв'язок між користувачами та їх ролями. Кожен користувач має відповідну роль, яка визначає рівень доступу до функцій системи. Наприклад, адміністратори мають повний доступ до всіх операцій у системі, включаючи блокування та деблокування карток, оновлення даних, а також доступ до детальної статистики. Оператори можуть лише працювати з картками, а експерти займаються верифікацією інформації та технічним обслуговуванням карток.

Модуль авторизації відповідає за реєстрацію користувачів у системі, їх аутентифікацію та присвоєння ролей. При реєстрації нового користувача в базу даних автоматично додаються його облікові дані, а також визначається його роль. Для кожного нового користувача система автоматично надає початкову роль «Оператор», яку адміністратор може змінити після перевірки.

Функціонал входу в систему забезпечує перевірку введених даних через процедуру аутентифікації. Після успішного входу користувач отримує відповідну роль, що дозволяє здійснювати доступ до лише тих функцій, які дозволені його роллю. Механізм перевірки прав доступу кожної ролі реалізований через функції контролю доступу, що зчитують дані про роль користувача з таблиці user_roles та визначають, чи дозволено виконання певних операцій.

Процес оновлення ролі користувача є важливою частиною функціоналу. Адміністратор має можливість змінювати роль користувача, наприклад, підвищуючи його до «Експерта» чи «Адміністратора». Ця операція здійснюється через спеціальний інтерфейс адміністратора, де відображається список користувачів та їх поточні ролі. Для змін в базі даних використовуються SQL-запити на оновлення значень таблиці user_roles.

Особливу увагу в модулі приділено безпеці. Для захисту даних від несанкціонованого доступу використовуються принципи шифрування та

перевірки прав доступу на кожному етапі роботи з системою. Усі запити до бази даних здійснюються через підготовлені SQL-запити, що мінімізує ризик SQL-ін'єкцій та інших видів атак. Крім того, для забезпечення високого рівня безпеки застосовано використання сесійних токенів для кожного користувача, що додатково захищає доступ до облікового запису.

Після аутентифікації користувач отримує роль, що визначає його доступ до таких функцій, як додавання нових карток, їх блокування, деблокування, оновлення інформації та інші операції, що вимагають адекватного рівня дозволу. Уся інформація про користувача та його ролі зберігається в базі даних, що забезпечує надійний та зручний механізм управління доступами.

Модуль ролей системи також забезпечує моніторинг дій користувачів та зберігання журналів їх активності. Завдяки цьому адміністрація системи може відслідковувати всі операції, здійснені користувачами, а також отримувати детальну інформацію про будь-які зміни, що відбулися в базі даних. Це є важливим для забезпечення прозорості роботи системи та контролю за її функціонуванням.

У результаті розробки цього модуля досягнуто високого рівня гнучкості та безпеки у використанні системи. Модуль дозволяє ефективно управляти правами доступу до різних функцій веб-додатку, забезпечуючи належний рівень контролю та захисту для кожної категорії користувачів.

3.4 Розробка модуля ремонту карток

У процесі реалізації функціональної частини веб-застосунку "CardFlow", особлива увага була приділена модулю ремонту карток, який забезпечує повний цикл обслуговування пошкоджених або несправних проїзних. Основною метою даного модуля є забезпечення ефективного та контрольованого процесу обробки заявок на ремонт, починаючи від моменту звернення користувача до остаточного вирішення проблеми з карткою.

Першим етапом реалізації є функція пошуку замовлення, що дозволяє оператору швидко знайти потрібну заявку за номером картки або іншим

унікальним ідентифікатором. Ця функція реалізована через форму з відповідним полем пошуку, яка звертається до бази даних і виводить усю необхідну інформацію про замовлення: ПІБ власника, тип картки, її статус, дата подачі заявки, наявність попередніх звернень. Для забезпечення зручності реалізовано можливість фільтрації за датами або статусами.

Наступним важливим елементом є функція ремонту, яка виконується після прийняття заявки в роботу. Цей процес передбачає перевірку технічного стану картки, фіксацію типу несправності (фізичне пошкодження, зчитування, програмний збій тощо) та реєстрацію відповідного рішення – відновлення або повна заміна. Оператор або експерт фіксує результати ремонту в системі, зазначаючи деталі виконаних дій, дату завершення робіт та відповідальну особу. Для уникнення помилок передбачено автоматичну валідацію даних, а також відображення спливаючих повідомлень у разі введення некоректної або неповної інформації.

Функція "в роботу" забезпечує переведення заявки зі статусу "Прийнята" до статусу "В роботі". Це дозволяє систематизувати черговість обслуговування, уникнути дублювання дій та забезпечити прозорий розподіл задач між працівниками. Після активації цієї функції до заявки додається позначка про початок робіт із фіксацією дати та часу, а також ідентифікатор працівника, який прийняв заявку. Додатково, система блокує можливість паралельної обробки одного й того ж замовлення іншими користувачами, що сприяє запобіганню конфліктів і втраті даних [14, с.190].

Інтегрованою частиною модуля є функція перегляду історії ремонту, що дозволяє отримати повну інформацію про всі попередні звернення щодо конкретної картки. Це включає дати ремонтів, опис проблем, здійснені дії, тривалість обслуговування та відповідальних працівників. Історія відображається у вигляді таблиці з можливістю сортування та експорту в формат Excel або PDF. Завдяки цьому підвищується прозорість сервісу, забезпечується контроль якості обслуговування, а також створюється база даних для подальшої аналітики.

Не менш важливою є функція перевірки даних, яка виконується автоматично при кожному етапі взаємодії з системою. Зокрема, перед внесенням або оновленням інформації система проводить перевірку на валідність даних – правильність формату номера картки, відсутність дублікатів, відповідність ПІБ базі реєстрації, актуальність терміну дії тощо. У разі виявлення помилок користувач отримує повідомлення з описом проблеми та рекомендаціями щодо її усунення. Це значно знижує ризики введення помилкових даних та забезпечує цілісність інформації у системі.

Таким чином, модуль ремонту карток є ключовим компонентом веб-застосунку "CardFlow", який дозволяє автоматизувати весь процес сервісного обслуговування, підвищити якість обробки звернень та забезпечити зручний контроль за виконанням заявок. Його реалізація сприяє ефективному функціонуванню сервісу, забезпечуючи прозорість, надійність та відповідність сучасним вимогам до інформаційних систем.

3.5 Висновок

Розділ 3 присвячений детальному аналізу та обґрунтуванню концептуальних підходів до реалізації веб-додатку для автоматизації обліку персональних проїзних карток у місті Вінниця. Після вивчення і оцінки різних аспектів, що стосуються розробки програмного забезпечення, можна зробити кілька важливих висновків, які підтверджують ефективність і доцільність запропонованої системи.

Одним із ключових аспектів розробки є ретельне планування архітектури додатку, що передбачає використання сучасних технологій і інструментів для забезпечення зручності в обслуговуванні карток. Застосування технології Flask для створення серверної частини дозволяє забезпечити високу продуктивність і зручність в обробці запитів.

Одним із основних результатів є розробка інтерфейсу, орієнтованого на простоту і зручність використання для адміністратора, оператора та експерта. Застосування сучасних технологій frontend-розробки, таких як HTML, CSS та

Phyton, дозволяє створити інтуїтивно зрозумілий і функціональний інтерфейс. Це суттєво зменшує час на навчання користувачів і забезпечує швидке освоєння системи.

Розроблений веб-застосунок передбачає чітке розмежування прав доступу між різними категоріями користувачів, що дозволяє забезпечити безпеку та контроль над усіма операціями в системі. Важливим аспектом є можливість адміністрування процесів блокування, деблокування та продовження терміну дії карток, що дозволяє максимально ефективно управляти ними та знижує ризик помилок через людський фактор.

Враховуючи специфіку обліку проїзних карток, особливо в контексті пільгових категорій населення, система надає можливість точного моніторингу термінів дії карток і пільг, що забезпечує зручність для користувачів і полегшує контроль за актуальністю даних. Система також дозволяє проводити ремонт карток, що є важливим для безперебійної роботи транспортної системи міста.

Одним з ключових моментів є використання принципу модульності при розробці програмного забезпечення. Це дозволяє системі легко масштабуватися в майбутньому, а також адаптуватися під нові вимоги без необхідності суттєвих змін у базовій архітектурі. Така гнучкість є важливим чинником у довгостроковій перспективі розвитку системи.

Завдяки ретельному тестуванню на етапі розробки було виявлено низку моментів, які дозволяють підвищити стабільність та ефективність роботи системи. Це включає в себе як внутрішнє тестування модулів, так і проведення тестування на реальних даних, що дозволяє переконатися в працездатності системи на практиці [15, с.156-158].

Система також враховує потреби зберігання і обробки великих обсягів даних, що є важливим для забезпечення стабільної роботи при значному збільшенні кількості користувачів та карток. Вона оптимізована для швидкої обробки запитів, що забезпечує високий рівень продуктивності навіть при великих навантаженнях.

Розроблений веб-застосунок відрізняється високим рівнем безпеки, що

забезпечується через використання сучасних методів аутентифікації та шифрування даних. Це гарантує захист особистої інформації користувачів та запобігає несанкціонованому доступу до системи.

На завершення, можна стверджувати, що запропонована система автоматизації обліку проїзних карток для міста Вінниця має високий потенціал для ефективного впровадження. Вона сприятиме покращенню якості обслуговування громадян, оптимізації процесів управління картками та забезпечить стабільність і безпеку роботи транспортної інфраструктури міста.

4. ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ ОБЛІКУ ПЕРСОНАЛЬНИХ ПРОЇЗНИХ КАРТОК

4.1 Тестування програмної системи

Тестування системи обліку та прийому персональних карток є критично важливим етапом у забезпеченні її стабільної роботи, відповідності функціональним вимогам та зручності використання для працівників і користувачів. Цей процес охоплює перевірку всіх основних компонентів системи, включаючи функціональність прийому карток, видачі, перевірки статусу, блокування та деблокування, обслуговування карток, адміністрування користувачів, а також аналіз логів. В цьому підрозділі буде детально розглянуто план тестування, методи, інструменти, які застосовувалися для перевірки, а також результати тестування системи.

Перевірка сторінки прийому карток є одним з ключових етапів тестування, адже вона відповідає за реєстрацію нових карток у системі. Основні аспекти, які тестувалися, включають правильність відображення полів введення, коректність обробки запитів на прийом, валідацію введених даних, відповідність логіки роботи загальному сценарію процесу прийому карток. На рисунку 4.1 представлено інтерфейс сторінки прийому карток.

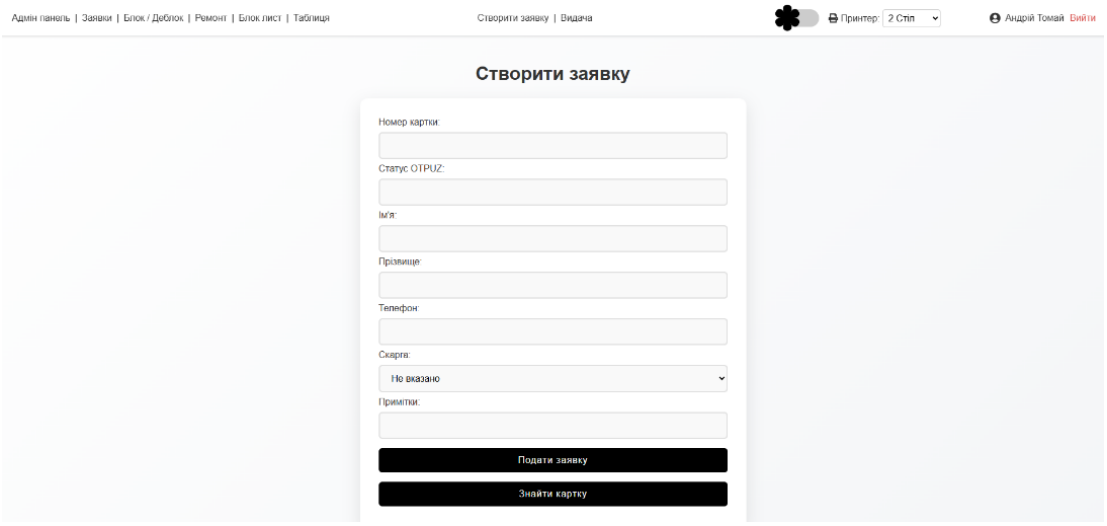

Рисунок 4.1 – Сторінка прийому карток

Важливо було перевірити, що система правильно обробляє всі випадки, включаючи сценарії, коли користувач вводить некоректні дані. Для цього проводилися негативні тестові випадки, де тестувалася поведінка системи при введенні неіснуючих або некоректних номерів карток.

Функціонал видачі карток дозволяє користувачам отримувати свої картки після ремонту або оформлення нових карток. Основне завдання тестування полягало у перевірці правильності відображення всіх елементів інтерфейсу, відповідності статусу картки реальному стану у базі даних та коректності оновлення даних після видачі.

На рисунку 4.2 показано сторінку видачі карток. Перевірка включала тестування поведінки системи при спробі видачі картки, яка ще не готова або не існує у базі даних. Важливо було підтвердити, що система відображає відповідні повідомлення про помилки і не дозволяє видавати картки, які ще перебувають у статусі ремонту або очікування.

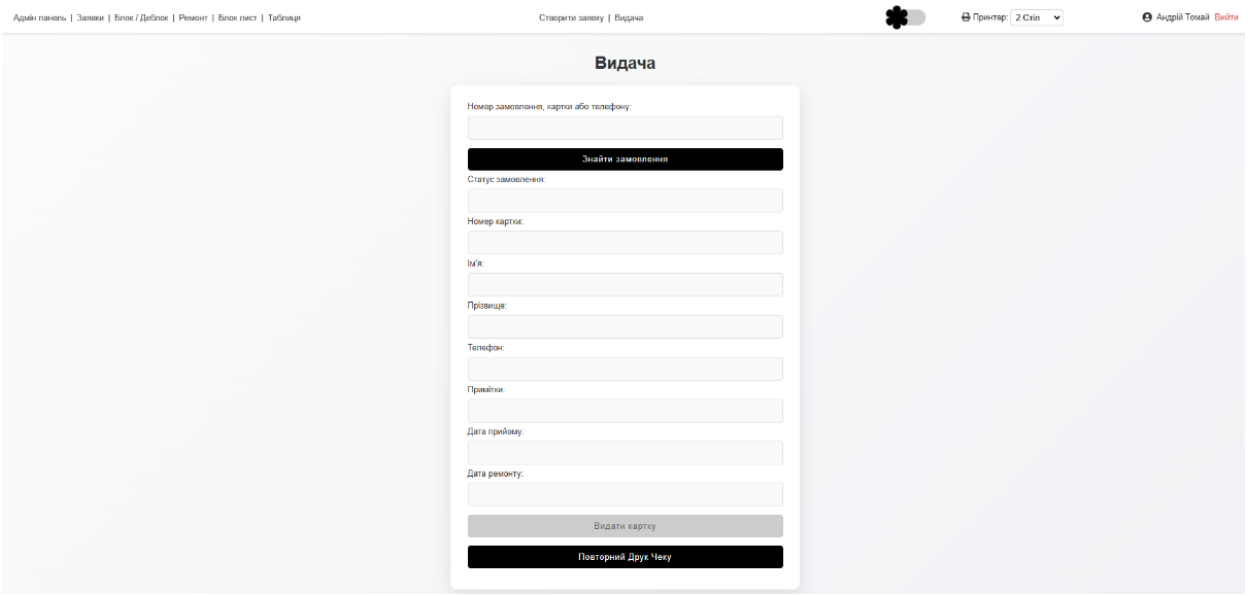

Рисунок 4.2 – Сторінка видачі карток

Система дозволяє користувачам переглядати інформацію про картку за певними критеріями, такими як дата заявки, статус замовлення, IP-адреса працівника, який прийняв картку, номер замовлення, дата ремонту, дата видачі

тощо. Це критично важливий функціонал, оскільки від його роботи залежить можливість отримання актуальних даних про картки.

Система дозволяє працівникам обробляти заявки на блокування або розблокування проїзних карток. На сторінці, зображеній на рисунку 4.3, представлена форма для введення номера картки, вибору дії (блокування або розблокування), зазначення причини та подання заявки. Це дозволяє швидко реагувати на звернення пасажирів, наприклад, у разі втрати картки чи виникнення технічних проблем.

У нижній частині сторінки реалізована функція перевірки вже поданих заявок. Користувач може здійснювати фільтрацію за номером картки, після чого в таблиці відображаються заявки з такими параметрами, як дата й час подання, номер картки, ім’я користувача, причина звернення, опис, статус обробки та підготовка. Під час тестування перевірялась коректність роботи фільтра, відповідність відображених даних записам у базі, а також логіка зміни статусів заявок.

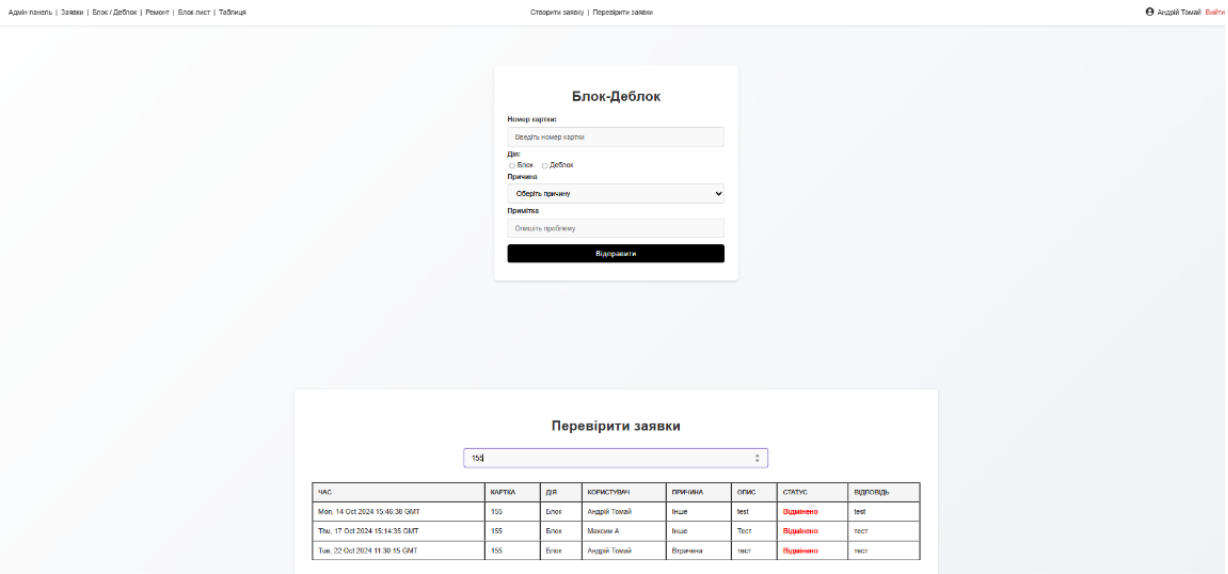

Рисунок 4.3 – Сторінка перевірки інформації по картці

Цей функціонал дозволяє блокувати картки, що були втрачені або пошкоджені, а також деблокувати їх після вирішення відповідних питань. Важливим етапом тестування було перевірити, що тільки користувачі з

відповідними правами можуть виконувати операції блокування або деблокування. На рисунку 4.4 представлена сторінка блокування/деблокування карток експертом з ремонту.

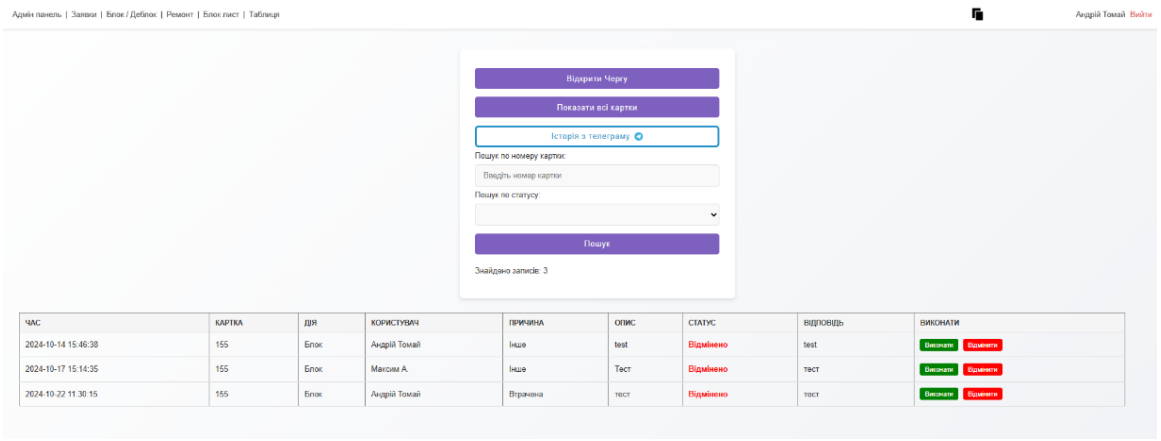

Рисунок 4.4 – Сторінка блокування та деблокування карток експертом з ремонту

Основні аспекти тестування включали перевірку правильності логіки блокування, оновлення статусу картки після виконання операції та коректність відображення повідомлень для користувачів.

Функціонал ремонту карток дозволяє спеціалістам змінювати статус картки на "В ремонті" або "Відремонтовано". Для перевірки цього функціоналу тестувалася відповідність статусів у базі даних фактичним змінам, що вносяться через інтерфейс.

На рисунку 4.5 відображено сторінку ремонту карток. Основне завдання тестування полягало у перевірці логіки роботи статусів та запису інформації по ремонту.

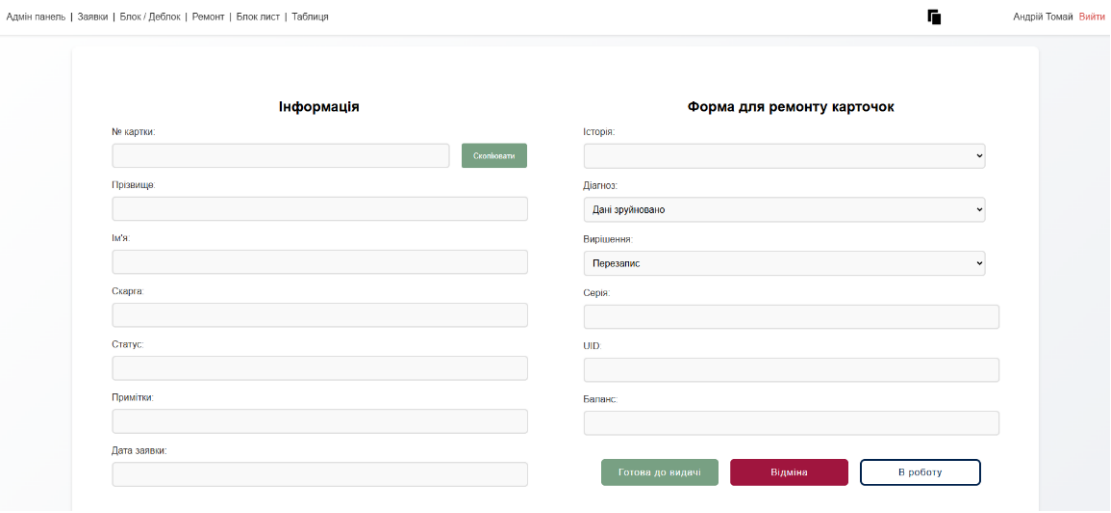

Рисунок 4.5 – Сторінка ремонту карток

Система передбачає можливість додавання, редагування та видалення користувачів з різними рівнями доступу. Важливим аспектом тестування було підтвердити, що система правильно обробляє запити на зміну ролей користувачів і не дозволяє виконувати дії, які виходять за межі дозволених прав. На рисунку 4.6 представлена сторінка управління користувачами.

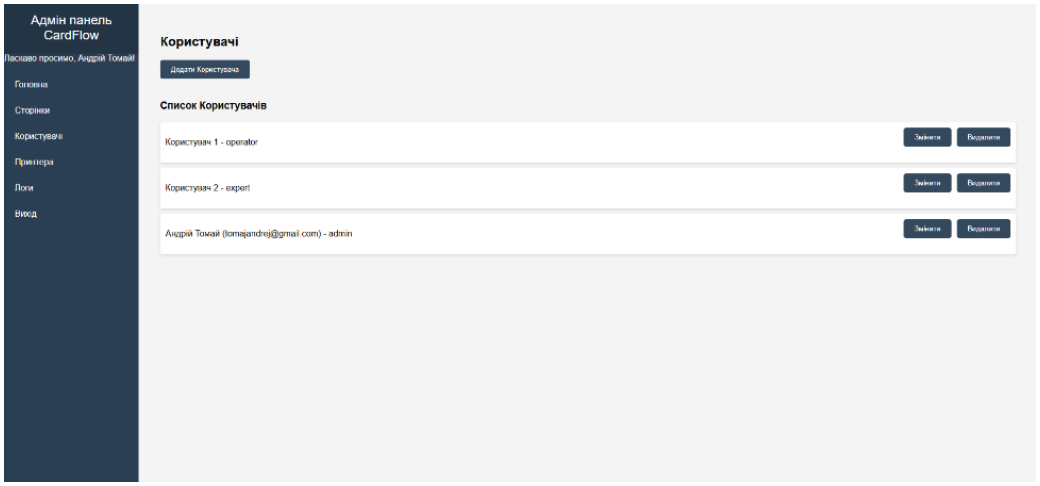

Рисунок 4.6 – Сторінка управління користувачами

Тестування включало перевірку, чи можна додати нового користувача з унікальним логіном, чи коректно працює зміна пароля та чи оновлюється інформація про користувача в базі даних після редагування.

Система веде детальний журнал дій користувачів, що дозволяє адміністраторам переглядати історію операцій. Це важливий функціонал для забезпечення прозорості роботи системи та можливості аудиту.

На рисунку 4.7 зображено сторінку перегляду логів. Тестування включало перевірку коректності записів у логах, можливість фільтрації за різними параметрами, а також перевірку доступності цієї інформації лише для користувачів з відповідними правами.

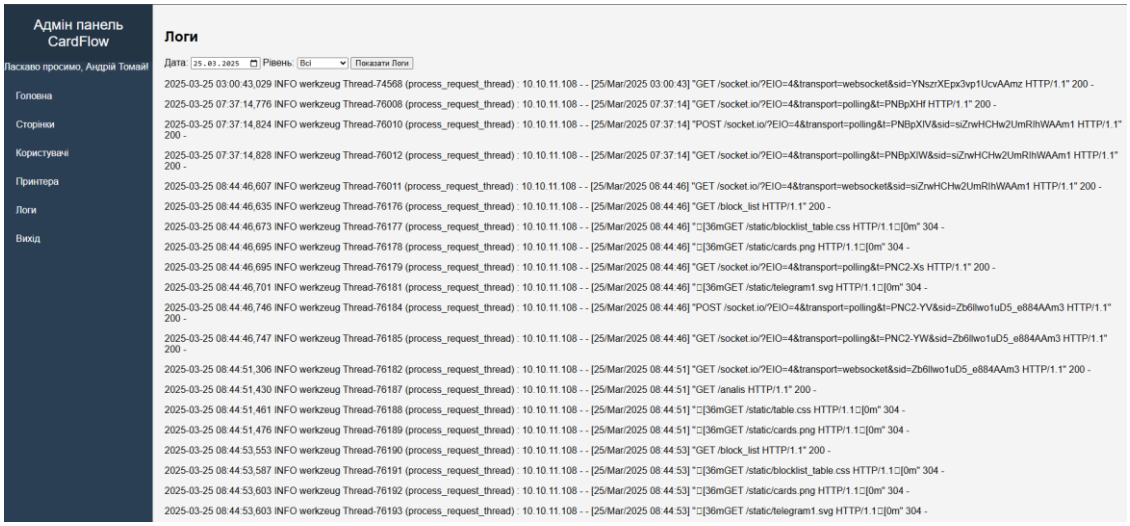

Рисунок 4.7 – Сторінка перегляду логів

Додатково у процесі тестування було проведено перевірку сторінки інформації по картках, яка містить історію операцій та дозволяє переглядати всі дії, що виконувалися з певною карткою. Ця сторінка є важливим елементом системи, оскільки вона забезпечує прозорість і контроль за всіма змінами, пов’язаними з картками. В рамках тестування перевірялася коректність відображення історії транзакцій, точність даних, відповідність записів у базі даних та зручність користування. Окремо було протестовано можливість фільтрації записів за датами, номерами карток, статусами та іншими критеріями. Також особливу увагу приділено швидкості завантаження великої кількості даних та їхньої коректної обробки при експортуванні у файл.

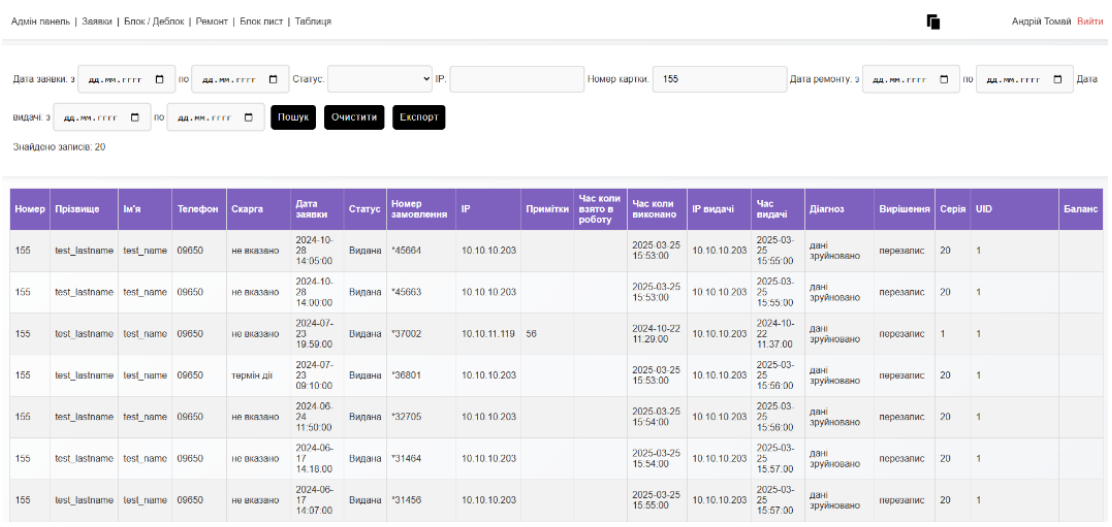

Рисунок 4.8 – Сторінка інформація по карткам

Ще одним важливим етапом тестування стало перевірка функції друку чеку звернення. Друк чеку є критично важливим для підтвердження виконаних дій та збереження інформації для клієнта чи адміністрації. У процесі тестування було перевірено коректність формування чеку, відображення всіх необхідних реквізитів, таких як номер замовлення, номер картки та ПІБ власника, дата прийому і скарга. Також тестувалися різні сценарії друку, включаючи перевірку сумісності з різними принтерами та форматами паперу.

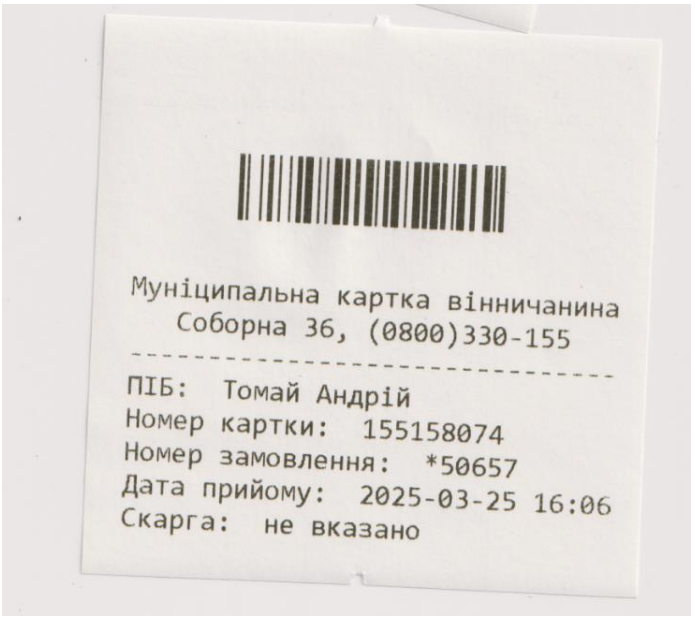

Рисунок 4.9 – Паперовий чек який отримує людина при зверненні

У результаті проведеного тестування було підтверджено стабільність і коректність роботи всіх функціональних модулів системи. Особливу увагу приділено валідації введених даних, перевірці коректності взаємодії з базою даних, а також швидкості виконання основних операцій. Виявлені під час тестування помилки були своєчасно виправлені, що дозволило покращити загальну продуктивність системи та підвищити рівень користувацького досвіду. Таким чином, система повністю відповідає поставленим вимогам і готова до повноцінного використання в реальному середовищі.

4.2 Створення інструкції для користувачів

Веб-застосунок для обліку та ремонту проїзних карток «CardFlow» був розроблений з метою забезпечення зручного і ефективного обслуговування користувачів різних категорій: клієнтів, операторів, експертів та адміністраторів. Метою даного веб-застосунку є спрощення та автоматизація процесів, пов'язаних з управлінням заявок на ремонт проїзних карток, а також забезпечення доступу до необхідної інформації кожному користувачеві відповідно до його ролі в системі.

Першим етапом роботи з веб-застосунком є авторизація користувача. Для того, щоб отримати доступ до системи, користувач повинен ввести свій логін та пароль. Крім того, для збереження безпеки і обмеження доступу тільки для користувачів, система вимагає, щоб користувач знаходився в межах локальної мережі підприємства. Цей підхід гарантує, що тільки співробітники, які мають відповідні повноваження, зможуть взаємодіяти з додатком.

Після успішної авторизації система надає доступ до функціоналу, що відповідає ролі користувача. Кожна роль має чітко визначені права доступу, що дозволяє обмежити доступ до певних функцій в залежності від того, чи є користувач клієнтом, оператором, експертом чи адміністратором.

Клієнти веб-застосунку «CardFlow» мають можливість подавати заявки на ремонт своїх проїзних карток. Після реєстрації запиту користувача клієнт

повинен надати основні дані, такі як номер картки, опис проблеми та прикріпити фотографії чи інші документи, що допоможуть у визначенні стану картки. Також передбачено введення контактної інформації, такої як номер телефону та адреса електронної пошти, для зв'язку з клієнтом.

Оператори веб-застосунку мають доступ до повного переліку поданих заявок на ремонт, включаючи деталі кожної заявки, інформацію про клієнта та додані документи. Оператори мають можливість змінювати статус заявки, відзначаючи її як «Прийнята», «Завершена» чи інші стани, в залежності від етапу обробки. Вони можуть переглядати інформацію про кожну заявку, зокрема опис проблеми, фотографії картки та інші супутні матеріали.

Експерти, у свою чергу, відповідають за безпосередню перевірку карток, які надійшли на ремонт. Після отримання заявки експерт проводить детальну діагностику картки, оцінюючи її технічний стан. На основі результатів перевірки експерт складає висновок, в якому вказує рекомендації щодо необхідності ремонту картки або її заміни.

Експерти можуть фіксувати результати перевірки безпосередньо у відповідних полях заявки в системі. Усі зафіксовані дані стають доступними для операторів та адміністраторів, що дозволяє здійснити подальші дії залежно від діагнозу картки, наприклад, призначення ремонту або відмови у ремонті через серйозні пошкодження.

Адміністратори веб-застосунку мають найвищі повноваження і доступ до всіх функцій системи. Вони можуть здійснювати управління ролями користувачів, встановлюючи рівень доступу для кожного користувача, в залежності від його функцій у системі. Адміністратори також можуть контролювати статистику заявок, переглядати звіти про кількість поданих заявок, їх статуси та інші важливі метрики, що дає змогу оцінювати ефективність роботи системи в цілому.

Крім того, адміністратори можуть налаштовувати сповіщення для різних подій в системі. Наприклад, можна налаштувати сповіщення про нові подані заявки, зміну статусу карток чи інші важливі події, що дозволяє тримати усіх

користувачів в курсі важливих змін. Адміністратори також відповідають за резервне копіювання даних, що гарантує безпеку і цілісність інформації.

Таким чином, веб-застосунок «CardFlow» забезпечує ефективну та зручну роботу для всіх категорій користувачів. Завдяки чітко структурованому інтерфейсу кожен користувач може швидко отримати доступ до необхідного функціоналу, а система ролей дозволяє забезпечити належний контроль доступу до різних функцій застосунку. Всі процеси автоматизовані, що значно зменшує час на обробку заявок та підвищує загальну ефективність роботи системи.

4.3 Визначення технічних вимог

Визначення технічних вимог є одним із ключових етапів у розробці веб-застосунку, оскільки воно забезпечує ефективну і безперебійну роботу системи на різних платформах. Для досягнення оптимальної продуктивності додатку необхідно правильно визначити параметри, які будуть відповідати як мінімальним вимогам для стартової роботи, так і рекомендованим для кращої взаємодії з користувачем. Окреме значення має врахування ресурсів серверної частини та клієнтської взаємодії через інтернет.

Нижче наведено технічні вимоги, які визначаються для роботи веб-додатку для обліку проїзних карток. Ці вимоги допоможуть забезпечити стабільну роботу додатку, зберігаючи при цьому можливість масштабування та адаптації до різних умов використання [16].

Таблиця 4.1 – Мінімальні технічні вимоги

Параметр	Вимога
Тип процесора	Двоядерний процесор 2.0 ГГц
Об’єм оперативної пам’яті	2 ГБ
Місце на жорсткому диску	150 МБ
Операційна система	Windows 7 / macOS 10.12 /Linux
Швидкість інтернету	1 Мбіт/с

Таблиця 4.2 – Рекомендовані технічні вимоги

Параметр	Вимога
Тип процесора	Чотириядерний процесор 3.0 ГГц
Об’єм оперативної пам’яті	4 ГБ
Місце на жорсткому диску	300 МБ
Операційна система	Windows 10 / macOS 11.0 / Linux
Швидкість інтернету	5 Мбіт/с

У випадку веб-додатків, процесор є критичним параметром для забезпечення швидкої обробки запитів і високої швидкості взаємодії з сервером. Для мінімальної роботи додатку достатньо двоядерного процесора з тактовою частотою 2.0 ГГц, що забезпечить швидку обробку основних операцій, таких як авторизація, перегляд карток і подання заявок. Однак для оптимальної роботи з більшими обсягами даних і багатозадачністю рекомендується використовувати чотириядерний процесор з тактовою частотою 3.0 ГГц, що дозволить зменшити затримки при обробці складніших запитів [17].

Оперативна пам’ять також є важливим чинником для забезпечення стабільності роботи веб-додатку. Мінімальний обсяг оперативної пам'яті 2 ГБ дозволяє додатку працювати з базовими функціями без значних затримок. Проте для комфортної роботи з великою кількістю відкритих вкладок та обробки більш складних процесів (запити на ремонт, перегляд історії заявок) рекомендується мати 4 ГБ оперативної пам'яті.

Для веб-додатку важливим є також обсяг вільного місця на жорсткому диску. Мінімальні 150 МБ дозволяють забезпечити зберігання необхідних даних, таких як інформація про користувачів, картки та заявки. Рекомендовані 300 МБ забезпечують достатній простір для тимчасових файлів, кешування та резервних копій, що дозволяє зберігати додаткові дані для покращення роботи користувача.

Щодо операційних систем, додаток підтримує найбільш популярні версії для веб-браузерів: Windows 10 або новіші версії, Linux а також macOS 10.12 і вище. Це забезпечує сумісність з більшістю персональних комп'ютерів, а також дає можливість стабільно працювати на сучасних браузерах, таких як Google Chrome, Mozilla Firefox, Microsoft Edge, Safari.

Швидкість інтернету також є важливою для забезпечення безперебійної роботи веб-додатку. Мінімальні вимоги – 1 Мбіт/с, що достатньо для базових операцій. Проте для комфортної роботи, включаючи швидке завантаження даних та відсутність затримок під час передачі інформації, рекомендується швидкість інтернету 5 Мбіт/с.

Визначення технічних вимог дає змогу забезпечити високу продуктивність веб-додатку, зменшити ймовірність технічних збоїв та покращити взаємодію користувача з системою.

4.4 Висновок

У розділі 4 було детально розглянуто процес розробки та технічного забезпечення веб-застосунку для обліку персональних проїзних карток, зокрема визначення основних вимог до його роботи та технічної інфраструктури. Визначення правильних технічних вимог є важливим кроком, оскільки вони безпосередньо впливають на ефективність та стабільність роботи застосунку, а також на його взаємодію з користувачами [18].

Для забезпечення комфортної роботи з додатком, особливо при великих обсягах даних та множинних запитах, важливо враховувати мінімальні та рекомендовані технічні вимоги, що дозволяють забезпечити стабільну та швидку роботу системи.

Веб-застосунок, як основний інструмент для користувачів, має бути оптимізований для швидкого доступу до даних та їх обробки в реальному часі. Це забезпечується за рахунок правильно налаштованої серверної частини та ефективного використання ресурсів клієнтської машини. У даному контексті важливим є вибір веб-технологій, що дозволяють зберігати високу швидкість роботи, зокрема, використання сучасних фреймворків і баз даних, які гарантують високу продуктивність та масштабованість системи.

Успішне визначення вимог до інтерфейсу та взаємодії з користувачем дозволяє створити зручний та інтуїтивно зрозумілий інтерфейс. Це, в свою чергу, дозволяє користувачам легко орієнтуватися в застосунку, без зайвих затримок та

помилок при виконанні основних функцій. Важливою частиною цього процесу є тестування на різних пристроях і операційних системах, що забезпечує безперебійну роботу веб-додатку для широкого кола користувачів.

Значну увагу варто приділяти оптимізації швидкості інтернет-з'єднання, оскільки це критично важливо для користувачів, які мають обмежену швидкість інтернету. Визначення оптимальних вимог до інтернет-з'єднання дозволяє забезпечити стабільну роботу застосунку при мінімальних затримках, а також зменшити ймовірність збоїв через погану якість з'єднання.

Не менш важливим є вибір програмного забезпечення та інструментів для розробки, які підтримують необхідні технічні вимоги і забезпечують легку масштабованість додатку в майбутньому. Технології, такі як Flask, Python, а також використання HTML та CSS для розробки фронтенду, дають змогу створити ефективний веб-додаток з мінімальними витратами на серверні ресурси [19, с.63].

У підсумку, визначення технічних вимог є основою для розробки веб-додатку, що відповідає потребам користувачів та забезпечує стабільну, безперебійну роботу в різних умовах експлуатації. Це дозволяє створити конкурентоспроможний продукт, який відповідає найвищим стандартам продуктивності, безпеки та зручності використання.

Технічні вимоги мають бути адаптовані до конкретних цілей проекту, з урахуванням якості зв'язку, типу користувачів та кількості оброблюваних даних. Успішне їх визначення на етапі планування дозволяє значно зменшити ймовірність проблем при запуску додатку і підвищити задоволеність користувачів [20].

Таким чином, розробка чітко визначених технічних вимог є основою для забезпечення високої якості та стабільності веб-додатку, що згодом визначить його успіх на ринку і буде сприяти зростанню кількості користувачів [21].

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було розглянуто та детально проаналізовано ключові аспекти, пов’язані з розробкою системи для обліку персональних проїзних карток у місті Вінниця. Завданням роботи було не лише розробити програмний продукт, але й визначити технічні вимоги, розробити структуру даних і запропонувати оптимальні рішення для забезпечення ефективної роботи системи.

Робота почалася з вивчення теоретичних основ та огляду сучасних підходів до створення веб-додатків. Основною метою було розробити систему, яка дозволяла б безпечно і зручно обробляти дані про користувачів та їх проїзні картки, а також забезпечити безперебійну роботу додатку на різних пристроях з урахуванням специфічних технічних вимог.

Однією з основних проблем, яку вирішував даний проект, було визначення вимог до апаратного та програмного забезпечення для забезпечення стабільної та швидкої роботи додатку. Для цього було проведено порівняння мінімальних та рекомендованих технічних вимог, що дало змогу визначити оптимальні параметри для різних типів пристроїв. Таке чітке визначення вимог гарантує правильне функціонування системи, зменшуючи ймовірність виникнення збоїв або неполадок під час її експлуатації.

Важливим етапом розробки стало визначення та оптимізація бази даних, що зберігає інформацію про користувачів, їх картки та операції. Структура даних була спроектована таким чином, щоб забезпечити високу швидкість обробки запитів і мінімізувати час відгуку при доступі до даних. Це є важливим аспектом, оскільки наявність великого обсягу інформації про користувачів та транзакції вимагає ефективного використання ресурсів системи.

Ще одним важливим аспектом роботи була розробка інтерфейсу веб-додатку. Оскільки система призначена для широкого кола користувачів, які можуть мати різний рівень технічної підготовки, інтерфейс повинен бути інтуїтивно зрозумілим та доступним. Було враховано зручність навігації,

легкість доступу до основних функцій та швидкість виконання операцій, що покращує користувацький досвід.

Веб-додаток був розроблений з урахуванням сучасних стандартів програмування та використанням популярних технологій, таких як Flask, Python, HTML і CSS. Використання цих інструментів дозволяє створити ефективний і гнучкий додаток, який можна швидко адаптувати до нових вимог та змін у середовищі. Також важливим аспектом було використання сучасних методів забезпечення безпеки, таких як шифрування даних та автентифікація користувачів, що гарантує захист інформації від несанкціонованого доступу.

Однією з найбільших складнощів на етапі реалізації стало забезпечення роботи додатку в умовах різних технічних характеристик користувачів, а також у ситуаціях з поганою якістю інтернет-з'єднання. Для цього було передбачено використання адаптивного дизайну та оптимізації зображень і контенту, що дозволяє зберігати ефективність роботи на пристроях з обмеженими ресурсами та в умовах низької швидкості інтернет-з'єднання.

Завдяки чітко визначеним технічним вимогам та правильно спроектованій архітектурі додаток продемонстрував високу продуктивність та зручність у використанні. Важливим досягненням є те, що створена система відповідає вимогам користувачів, забезпечуючи зручний доступ до даних та швидке виконання основних функцій.

Також необхідно відзначити, що робота включає проведення серії тестувань, що дозволило виявити й усунути основні недоліки, а також оптимізувати роботу системи. Це є важливою частиною процесу розробки, оскільки дозволяє забезпечити стабільну роботу додатку в реальних умовах експлуатації. Врахування результатів тестувань дозволяє значно підвищити надійність програмного продукту.

В цілому, робота продемонструвала важливість чіткого визначення технічних вимог, правильного вибору технологій та ретельного тестування на етапі розробки. Завдяки цим аспектам вдалося створити систему, яка задовольняє вимоги користувачів і ефективно виконує основні функції.

У ході виконання роботи були отримані не тільки практичні результати, а й важливі теоретичні знання щодо процесу розробки систем та управління проектами.

Отже, виконана бакалаврська кваліфікаційна робота є успішною розробкою системи, яка задовольняє сучасні вимоги до функціональності, безпеки та зручності використання. Результати дослідження можуть бути використані як основа для подальших розробок у цій галузі та для вдосконалення існуючих систем обліку персональних карток у містах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Альтернатива CRM Gincore - RemOnline URL: https://remonline.ua/remonline-vs-gincore/ (дата звернення: 14.04. 2025).

2. Порівняння Gincore та RemOnline URL: https://remonline.ua/remonline-vs-gincore/ (дата звернення: 14.04. 2025).

3. ЛеоКарт URL: https://leocard.lviv.ua/ (дата звернення: 14.04. 2025).

4. Розробка облікових програм на Python URL: https://avada-media.ua/services/razrabotka-uchetnykh-programm-na-python/ (дата звернення: 14.04. 2025).

5. HTML і CSS: що це, кому та для чого потрібно URL: https://goit.global/ua/articles/html-i-css-shcho-tse-komu-ta-dlia-choho-potribno/ (дата звернення: 14.04. 2025).

6. Чому блокуються банківські картки вінничан? URL: https://vn.20minut.ua/DTP/nemozhlivo-rozrahuvatisya-za-proyizd-chomu-blokuyutsya-bankivski-kartk-11946134.html (дата звернення: 14.04. 2025).

7. Яременко Л. М., Пономаренко Я. А. Сучасні тенденції впровадження управлінського обліку із застосуванням міжнародного досвіду. Вісник Житомирського державного технологічного університету. Серія: Економіка, управління та адміністрування. 2019. № 2 (88). С. 144-148.

8. Биба В. В., Матюшіна Ю. І. Система управлінського обліку: сутність, завдання та етапи впровадження. Економіка та держава. 2015. № 1. С. 60-62.

9. Задорожний З.-М. В., Аверкин Я. Ф. Управлінський облік: особливості та принципи. Фінансовокредитна діяльність: проблеми теорії та практики. 2019. №1. С. 114-120

10. PyCharm: the Python IDE for data science URL: https://www.jetbrains.com/pycharm/ (дата звернення: 14.04. 2025).

11. Visual Studio Code - Code Editing. Redefined URL: https://code.visualstudio.com/ (дата звернення: 14.04. 2025).

12. Sublime Text 3 URL: https://www.sublimetext.com/3 (дата звернення:

14.04. 2025).

13. Обзор редактора Atom URL: https://itproger.com/ua/course/programming/22 (дата звернення: 14.04. 2025).

14. М'якшило, О. М. CASE-технології у проектуванні інформаційних систем: навч. посіб. / О. М. М'якшило, Л. Г. Загоровська. — Київ : НУХТ, 2017. — 190 с.

15. Задорожнюк Н.О. Сучасне програмне забезпечення для здійснення бізнесаналізу. Економічний вісник НТУУ «Київський політехнічний інститут». 2021. № 19. с. 156-158.

16. ТОП 10 вимог до сучасного веб-сайту URL: https://laweb.com.ua/uk/blog/top-10-vimog-do-suchasnogo-veb-saytu (дата звернення: 14.04. 2025).

17. Правила та умови використання веб-сайту URL: https://ux.pub/terms#google_vignette (дата звернення: 14.04. 2025).

18. Біліченко В. В. Аналіз проблем при впровадженні єдиного електронного квитка на громадському транспорті. URL: http://ir.lib.vntu.edu.ua/bitstream/handle/123456789/21900/material2018-25-27.pdf?sequence=1 (дата звернення: 14.04. 2025).

19. Томай А.О., Войтко В.В., Рельке А.А., Байдалюк.В.В Розробка програмного додатку для управління пристроями користувача. Електронні інформаційні ресурси: створення, виконання, доступ в зб: Міжнародної науково-практичної Інтернет-конференції наук.-практ. конф. Вінниця: Полісся, 2022. С.63.

20. Томай А.О. Кобилянська І. М. Правове регулювання охорони праці: обов'язки роботодавців та права працівників. Молодь в науці: дослідження, проблеми, перспективи (МН-2024).

21. Томай А.О. Бабюк Н.П. Функціональні особливості проїзних карток. Молодь в науці: дослідження, проблеми, перспективи (МН-2025).

63

ДОДАТКИ

Додаток А. Технічне завдання

Міністерство освіти та науки України

Вінницький національний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Директор КП «ВІЦ»
С.С Бригадир
«25» березня 2025р

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.н.т., професор
О.Н.Романюк
«25» березня 2025р.

Технічне завдання

на бакалаврську кваліфікаційну роботу «Розробка програмної системи обліку персональних проїзних карток м. Вінниці» за спеціальністю 121-Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н, доцент Н.П.Бабюк

Виконав:
студент гр. А.О.Томай
«24» березня 2025р.

Вінниця-2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка програмної системи обліку персональних проїзних карток м. Вінниці».

Галузь застосування – веб-застосунок.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від «20» березня 2025 р. Ректора ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою бакалаврської кваліфікаційної роботи є розробка веб-додатку для обліку персональних проїзних карток у місті Вінниця, який забезпечить ефективну організацію обліку, зручний доступ до інформації та покращення якості надання послуг користувачам. Додаток має на меті оптимізувати процеси реєстрації, перевірки та використання проїзних карток, забезпечуючи при цьому безпеку даних і зручність користування.

Призначення роботи – розробка та реалізація веб-додатку для обліку персональних проїзних карток, який дозволить автоматизувати процеси, пов'язані з управлінням картками, а також надасть користувачам доступ до основних функцій, таких як перевірка балансу, поповнення картки та перегляд історії транзакцій. Система повинна бути зручною як для користувачів, так і для адміністрації, забезпечуючи швидкий доступ до даних і високий рівень надійності.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР:

1. М'якшило, О. М. CASE-технології у проектуванні інформаційних систем: навч. посіб. / О. М. М'якшило, Л. Г. Загоровська. — Київ : НУХТ, 2017.

— 190 с.

2. Задорожнюк Н.О. Сучасне програмне забезпечення для здійснення бізнесаналізу. Економічний вісник НТУУ «Київський політехнічний інститут». 2021. № 19. с. 156-158.

3. ТОП 10 вимог до сучасного веб-сайту URL: https://laweb.com.ua/uk/blog/top-10-vimog-do-suchasnogo-veb-saytu (дата звернення: 14.04. 2025).

4. Правила та умови використання веб-сайту URL: https://ux.pub/terms#google_vignette (дата звернення: 14.04. 2025).

5. Біліченко В. В. Аналіз проблем при впровадженні єдиного електронного квитка на громадському транспорті. URL: http://ir.lib.vntu.edu.ua/bitstream/handle/123456789/21900/material2018-25-27.pdf?sequence=1 (дата звернення: 14.04. 2025).

6. Томай А.О., Войтко В.В., Рельке А.А., Байдалюк.В.В Розробка програмного додатку для управління пристроями користувача. Електронні інформаційні ресурси: створення, виконання, доступ в зб: Міжнародної науково-практичної Інтернет-конференції наук.-практ. конф. Вінниця: Полісся, 2022. С.63.

7. Томай А.О., Кобилянська І. М. Правове регулювання охорони праці: обов'язки роботодавців та права працівників. Молодь в науці: дослідження, проблеми, перспективи (МН-2024).

8. Томай А.О. Бабюк Н.П. Функціональні особливості проїзних карток. Молодь в науці: дослідження, проблеми, перспективи (МН-2025).

5. Технічні вимоги

Тип програми – веб-додаток; модель розробки – ітеративна; засоби зберігання даних – Firebase; вхідні дані: персональні дані користувача, дані про проїзні картки, історія транзакцій; вихідні дані: інформація про баланс картки, історія використання картки, звіт про оплату проїзду; середовище розробки – PyCharm, Flask, HTML, CSS; мови програмування – Python (Flask), HTML, CSS;

програмне забезпечення для тестування веб-додатків – Chrome Developer Tools.

У межах цієї роботи розроблено веб-додаток для обліку персональних проїзних карток у місті Вінниця.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

1. Пояснювальна записка до БКР.
2. Технічне завдання.
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану системи обліку персональних проїзних карток у сучасному світі	25.03.25 – 01.04.25
2	Аналіз аналогів програмної системи обліку персональних проїзних карток	01.04.25 – 08.04.25
3	Розробка алгоритмів системи	08.04.25 – 29.04.25
4	Розробка модулів системи	29.04.25 – 20.05.25
5	Тестування системи	20.05.25 – 28.05.25
6	Оформлення матеріалів до захисту БКР	28.05.25 – 01.06.25

10. Порядок контролю та прийняття

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

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

68

Додаток Б. Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмної системи обліку персональних проїзних карток м. Вінниці

Тип роботи: Бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра ПЗ, ФІТКІ, 5ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 3 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Бабюк А.П., к.т.н., доцент
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Тимч А. О.
(підпис) (прізвище, ініціали)

Додаток В. Акт впровадження

УЗГОДЖЕНО
Директор КП "Вінницький інформаційний центр"
С.С. Бригадир
«28» травня 2025 року

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор О.Н. Романюк
«28» травня 2025 року

АКТ ВПРОВАДЖЕННЯ № 1
результатів науково-дослідних робіт

Замовник КП "Вінницький інформаційний центр"
(найменування організації)

Цим актом підтверджується, що результати роботи – Розробка програмної системи обліку персональних проїзних карток м.Вінниці.
(найменування теми)

що виконав студент гр. 5ПІ – 21б ВНТУ Томай А.О.
(виконавець)

за трудовим договором з КП «ВІЦ», на посаді Оператора з обробки інформації та програмного забезпечення відділу інформаційних технологій та системного адміністрування виконана з 25.03.2025 по 28.05.2025
(строки виконання)

впроваджено у Комунальному підприємстві "Вінницький інформаційний центр"
(найменування організації, де здійснювалося впровадження)

1. Вид впроваджених результатів: експлуатація програмного забезпечення
(експлуатація виробу, роботи, технології)
2. Характеристика масштабу впровадження: одиничне
(унікальне, одиничне, партія, масове, серійне)
3. Форма впровадження: дослідний зразок
4. Новизна результатів науково-дослідної роботи: якісно нові
(піонерські, принципово нові, якісно нові, модифікації, модернізація старих розробок)
5. Впроваджені: у відділі інформаційних технологій та системного адміністрування
6. Соціальний та науково-технічний ефект: спеціальне призначення
(охорона навколишнього середовища, поліпшення й оздоровлення умов праці, удосконалення структури керування, науково-технічних напрямків, спеціальне призначення)

Від виконавця:
студент групи 5ПІ-21Б
А. О. Томай
Керівник: к.т.н., доцент кафедри ПЗ

Від КП "Вінницький інформаційний центр
С. С. Бригадир
Н. П. Бабюк

Додаток Г. Лістинг програми

App.py

```
from functools import wraps
from flask import render_template_string, render_template, redirect, url_for, flash
from flask_login import LoginManager, UserMixin, login_user, login_required,
logout_user, current_user
from flask import Flask, request, jsonify
import time
import Admin_panel
import Users
from werkzeug.security import check_password_hash
import printer
import traceback
import threading
import Postgresql
from configparser import ConfigParser
import csv
from io import StringIO
from flask import make_response
import logging
from datetime import timedelta
import datetime
import re
from flask_socketio import SocketIO
current_date = datetime.datetime.now().date()
logging.basicConfig(filename=f'app.log', level=logging.DEBUG,
                    format='%(asctime)s %(levelname)s %(name)s %(threadName)s :
%(message)s')

config = ConfigParser()
config.read('config.ini')
app = Flask(__name__)
app.secret_key = config['DEFAULT']['APP_SECRET_KEY']
app.config['REMEMBER_COOKIE_DURATION'] = timedelta(days=30)

login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'
socketio = SocketIO(app)


# User class
class User(UserMixin):
   def __init__(self, id, email, username, role):
      self.id = id
      self.email = email
      self.username = username
      self.role = role
```

```
        def role_required(*roles):
            try:
                def decorator(f):
                    @wraps(f)
                    def decorated_function(*args, **kwargs):
                        if not current_user.is_authenticated or current_user.role not in roles:
                            flash('Відмовлено в доступі', 'error')
                            return redirect(url_for('login'))
                        return f(*args, **kwargs)

                    return decorated_function

                return decorator
            except Exception as e:
                logging.error(f'Помилка role_required: {e}')
                print("Помилка:", e)


        @login_manager.user_loader
        def load_user(user_id):
            try:
                user_data = Users.get_user_by_id(user_id)
                if user_data:
                    return User(id=user_data['id'], email=user_data['email'],
username=user_data['username'],
                                role=user_data['role'])
                return None
            except Exception as e:
                logging.error(f'Помилка load_user: {e}')
                print("Помилка:", e)
                return None


        @app.route('/login', methods=['GET', 'POST'])
        def login():
            try:
                if request.method == 'POST':
                    email = request.form['email']
                    password = request.form['password']
                    remember_me = request.form.get('remember-me')
                    next_url = request.form.get("next")
                    user_data = Users.get_user(email)
                    if user_data and check_password_hash(user_data['password'], password):
                        user = User(id=user_data['id'], email=user_data['email'],
username=user_data['username'],
                                    role=user_data['role'])
                        login_user(user, remember=remember_me)
                        if next_url:
                            logging.info(f'Користувач: {user_data["username"]} увійшов')
                            return redirect(next_url)
                        return redirect(url_for('home'))
```

```
            else:
                flash('Невірний логін або пароль', 'error')
                if next_url:
                    return redirect(next_url)
                return redirect(url_for('login'))
        return render_template('login.html')
    except Exception as e:
        print("Помилка:", e)
        logging.error(f'Помилка login: {e}')
        return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500


@app.route('/logout')
@login_required
def logout():
    try:
        logout_user()
        return redirect(url_for('login'))
    except Exception as e:
        print("Помилка:", e)
        logging.error(f'Помилка logout: {e}')
        return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500


@app.route('/admin_panel', methods=['GET', 'POST'])
@login_required
@role_required('admin')
def admin_panel():
    try:
        username = current_user.username
        statistic = Admin_panel.counter()

        return render_template('admin_panel.html', username=username, statistic=statistic)
    except Exception as e:
        print("Помилка:", e)
        logging.error(f'Помилка admin panel: {e}')
        return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500


@app.route('/get_logs', methods=['GET'])
@login_required
@role_required('admin')
def get_logs():
    try:
        date = request.args.get('date')
        level = request.args.get('level')

        with open('app.log', 'r') as file:
            logs = file.readlines()

        if date:
            date_pattern = re.compile(rf'^{date}.*')
```

```
                logs = [log for log in logs if date_pattern.match(log)]

            # Filter logs by level
            if level:
                level_pattern = re.compile(rf'.*\b{level}\b.*')
                logs = [log for log in logs if level_pattern.match(log)]

            return jsonify(logs)
        except Exception as e:
            print("Помилка:", e)
            logging.error(f'Помилка get_logs: {e}')
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500


@app.route('/get_chart_data', methods=['GET'])
@login_required
@role_required('admin')
def get_chart_data_endpoint():
    added_data = Admin_panel.get_chart_data()
    repair_data = Admin_panel.get_chart_data_repair()
    issue_data = Admin_panel.get_chart_data_issue()

    combined_data = {}
    for date, count in added_data:
        combined_data[date] = [max(count, 0), 0, 0]  # Ensuring non-negative values
    for date, count in repair_data:
        if date in combined_data:
            combined_data[date][1] = max(count, 0)
        else:
            combined_data[date] = [0, max(count, 0), 0]
    for date, count in issue_data:
        if date in combined_data:
            combined_data[date][2] = max(count, 0)
        else:
            combined_data[date] = [0, 0, max(count, 0)]

    final_data = [["Дата", "Прийнято", "Зроблено", "Видано"]]
    for date, counts in sorted(combined_data.items()):
        final_data.append([date.strftime('%d-%m'), *counts])

    return jsonify(final_data)


@app.route('/add_user', methods=['POST'])
@login_required
@role_required('admin')
def add_user():
    try:
        Users.add_user(request.form['username'], request.form['email'],
request.form['password'], request.form['role'],
                       None)
        flash('Користувача додано успішно!', 'success')
```

```
            logging.info(f'Користувач {request.form["username"]} доданий')
            return redirect(url_for('admin_panel'))
        except Exception as e:
            print("Помилка:", e)
            flash('Сталася внутрішня помилка сервера.', 'error')
            logging.error(f'Помилка add user: {e}')
            return redirect(url_for('admin_panel'))


@app.route('/delete_user', methods=['POST'])
@login_required
@role_required('admin')
def delete_user():
    data = request.get_json()
    email = data.get('email')
    try:
        Users.delete_user(email)
        logging.info(f'Користувач {email} видалений')
        return jsonify({'success': True})
    except Exception as e:
        print("Помилка:", e)
        logging.error(f'Помилка delete user: {e}')
        return jsonify({'success': False, 'message': 'Сталася внутрішня помилка сервера.'})


@app.route('/get_users', methods=['GET'])
@login_required
@role_required('admin')
def get_users():
    try:
        users = Users.list_users()
        return jsonify(users)
    except Exception as e:
        print("Помилка:", e)
        logging.error(f'Помилка get users: {e}')
        return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500


@app.route('/get_user', methods=['GET'])
@login_required
@role_required('admin')
def get_user():
    email = request.args.get('email')
    if not email:
        return jsonify({'error': 'Email parameter is required'}), 400

    try:
        user = Users.get_user_by_email(email)
        if user:
            return jsonify(user)
        else:
            return jsonify({'error': 'User not found'}), 404
```

```
            except Exception as e:
                print("Помилка:", e)
                logging.error(f'Помилка get user: {e}')
                return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500


        @app.route('/edit_user', methods=['POST'])
        @login_required
        @role_required('admin')
        def edit_user():
            try:
                Users.update_user(request.form['email'], request.form['username'],
request.form['password'],
                                  request.form['role'])
                flash('Дані користувача змінено успішно!', 'success')
                logging.info(f'Користувач {request.form["username"]} змінений')
                return redirect(url_for('admin_panel'))
            except Exception as e:
                print("Помилка:", e)
                flash('Сталася внутрішня помилка сервера.', 'error')
                logging.error(f'Помилка edit user: {e}')
                return redirect(url_for('admin_panel'))


        @app.route('/', methods=['GET', 'POST'])
        @login_required
        def home():
            username = current_user.username
            userrole = current_user.role

            return render_template('form.html', username=username, role=userrole)


        @app.route('/update_printer', methods=['POST'])
        @login_required
        def update_printer():
            try:
                data = request.get_json()
                printer_value = data.get('printer')
                username = current_user.username
                user_id = current_user.id
                Users.update_user_config(user_id, printer_value)

                print(f'Встановлено : {printer_value} для {username} ({user_id})')
                logging.info(f'Встановлено : {printer_value} для {username} ({user_id})')
                return jsonify({'success': True})
            except Exception as e:
                print("Помилка:", e)
                logging.error(f'Помилка update_printer: {e}')


        @app.route('/get_printer', methods=['POST'])
```

```
@login_required
def get_printer():
    try:
        user_id = current_user.id
        printer_name = Users.get_user_config(user_id)
        return jsonify({'success': True, 'printer': printer_name})
    except Exception as e:
        print("Помилка:", e)
        logging.error(f'Помилка get_printer: {e}')
        return jsonify({'success': False, 'error': 'Сталася внутрішня помилка сервера.'}),
def check_print_name(user_id):
    printer_name = Users.get_user_config(user_id)
    if printer_name == 'printer1':
        return config['DEFAULT']['PRINTER_1_Table']
    elif printer_name == 'printer2':
        return config['DEFAULT']['PRINTER_2_Table']
    elif printer_name == 'printer3':
        return config['DEFAULT']['PRINTER_3_Table']
    elif printer_name == 'printer4':
        return config['DEFAULT']['PRINTER_4_Table']
    elif printer_name == 'printer5':
        return config['DEFAULT']['PRINTER_5_Table']
    else:
        return config['DEFAULT']['PRINTER_NAME']


def print_async(printer_path, order_number, card_number, first_name, last_name,
phone_number, date, complaint):
    try:
        # Ваш код для друку
        printer.print_check(printer_path, order_number, card_number, first_name,
last_name, phone_number, date,
                            complaint)
    except Exception as e:
        print("Помилка під час друку:", e)
        logging.error(f'Помилка під час друку print_async: {e}')
        traceback.print_exc()


@app.route('/send_card_sheets', methods=['POST'])
@login_required
def send_card_sheets():
    try:
        card_number = request.form['card_number']
        first_name = request.form['first_name']
        last_name = request.form['last_name']
        phone_number = request.form['phone']
        complaint = request.form['complaint']
        user_id = current_user.id
        user_ip = request.remote_addr
        user_name = current_user.username
        notes = request.form.get('notes', '')
```

```
            if not card_number or len(card_number) > 9:
                return jsonify({'error': 'Перевірте номер карти'})

            # Перевірка чи картка вже зареєстрована
            order = Postgresql.search_order(card_number)
            if order and order.get('status') == 'Прийнята':
                return jsonify({'error': 'Картка вже зареєстрована'})


            order_number = '*' + str(Postgresql.get_order_number())
            print('Запис в POSTGRESQL -', card_number, last_name, first_name,
phone_number, complaint,
                  time.strftime("%Y-%m-%d %H:%M"), order_number, user_name, notes)
            Postgresql.add_orders(card_number, last_name, first_name, phone_number,
complaint,
                                  time.strftime("%Y-%m-%d %H:%M"), 'Прийнята', order_number,
user_name, notes)
            logging.info(
                f"Запис в POSTGRESQL - {card_number}, {last_name}, {first_name},
{phone_number}, {complaint}, {time.strftime('%Y-%m-%d %H:%M')}, {order_number},
{user_name}, {notes}"
            )
        except Exception as e:
            print("Сталася помилка:", e)
            logging.error(f'Помилка send_card_sheets: {e}')
            traceback.print_exc()
            return jsonify({'success': False, 'message': 'Сталася внутрішня помилка
сервера.'}), 500
        print_thread = threading.Thread(target=print_async, args=(
            check_print_name(user_id), order_number, card_number, first_name, last_name,
phone_number,
            time.strftime("%Y-%m-%d %H:%M"), complaint,))
        print_thread.start()
        return jsonify({'success': True})


    @app.route('/get_card_info', methods=['POST'])
    @login_required
    def get_card_info():
        try:
            card_number = request.form['card_number']
            if not card_number:
                return jsonify({'error': 'Введіть номер карти'})
            data = Postgresql.search_card(card_number)
            if data:
                first_name = data[3]
                last_name = data[2]
                phone = data[5]
                status_otpus = data[7]
                logging.info(f'Користувач {current_user.username} отримав дані про картку
{card_number}')
                return jsonify({'first_name': first_name, 'last_name': last_name, 'phone': phone,
```

```
'statusot': status_otpus})

            return jsonify({'error': 'Дані для цьої картки в OTPUZ не знайдено'})
        except Exception as e:
            print("Помилка:", e)
            logging.error(f'Помилка get_card_info: {e}')
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500


    @app.route('/find_order', methods=['POST'])
    @login_required
    def find_order():
        try:
            order_number = request.form.get('order_number')
            if not order_number:
                return jsonify({'error': 'Введіть номер карти'})

            order_data = Postgresql.search_order(order_number)

            if order_data:
                return jsonify({'success': True, 'order': order_data})
            else:
                return jsonify({'success': False})
        except Exception as e:
            print("Помилка:", e)
            logging.error(f'Помилка find_order: {e}')
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500


    @app.route('/get_dates', methods=['POST'])
    @login_required
    def get_dates():
        try:
            card_number = request.form.get('card_number')
            if not card_number:
                return jsonify({'error': 'Введіть номер карти'})
            dates = Postgresql.get_dates(card_number)
            return jsonify(dates)
        except Exception as e:
            print("Помилка:", e)
            logging.error(f'Помилка get_dates: {e}')
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500


    @app.route('/get_history', methods=['POST'])
    @login_required
    def get_history():
        try:
            card_number = request.form.get('card_number')
            time_add = request.form.get('time_add')
            if not card_number:
                return jsonify({'error': 'Введіть номер карти'}), 400
```

```
            history = Postgresql.get_history(card_number, time_add)

            return jsonify(history)
        except Exception as e:
            print("Помилка:", e)
            logging.error(f'Помилка update_printer: {e}')
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500


    @app.route('/card_to_work', methods=['POST'])
    @login_required
    def card_to_work():
        try:
            card_number = request.form.get('card_number')
            order_date = request.form.get('order_date')
            notes = request.form.get('notes')
            if not card_number:
                return jsonify({'error': 'Введіть номер карти'}), 400

            success = Postgresql.card_to_work(card_number, order_date, notes)

            if success:
                return jsonify({'success': True})
            else:
                return jsonify({'success': False, 'message': 'Не вдалося оновити статус.'})
        except Exception as e:
            print("Помилка:", e)
            logging.error(f'Помилка card_to_work: {e}')
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500


    @app.route('/set_history', methods=['POST'])
    @login_required
    def set_history():
        try:
            card_number = request.form.get('card_number')
            diagnosis = request.form.get('diagnosis')
            complite_com = request.form.get('complite_com')
            series = request.form.get('series')
            uid = request.form.get('uid')
            time_add = request.form.get('time_add')
            balance = request.form.get('balance')
            notes = request.form.get('notes')

            if not card_number or not diagnosis or not complite_com or not series or not uid or
not time_add:
                return jsonify({'error': 'Заповніть всі дані.'})

            success = Postgresql.set_history(card_number, diagnosis, complite_com, series, uid,
balance, notes, time_add,
                                             time.strftime("%Y-%m-%d %H:%M"))
```

```
            print('Запис даних про ремонт в POSTGRESQL -', card_number, diagnosis,
complite_com, series, uid, balance,
                  notes, time_add, time.strftime("%Y-%m-%d %H:%M"))
            logging.info(
               f"Запис даних про ремонт в POSTGRESQL - {card_number}, {diagnosis},
{complite_com}, {series}, {uid}, {balance}, {notes}, {time_add}, {time.strftime('%Y-%m-%d
%H:%M')}"
            )
            if success:
               return jsonify({'success': True})
            else:
               return jsonify({'success': False, 'message': 'Не вдалося оновити історію.'})
         except Exception as e:
            print("Помилка:", e)
            logging.error(f'Помилка set_history: {e}')
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500


      @app.route('/print_check', methods=['POST'])
      def print_check():
         try:
            order_number = request.form.get('order_number')
            if not order_number:
               return jsonify({'error': 'Введіть номер карти'})
            print_data = Postgresql.search_order(order_number)
            if not print_data:
               return jsonify({'error': 'Замовлення не знайдено.'})
            print_thread = threading.Thread(target=print_async, args=(
               check_print_name(user_id=current_user.id), print_data['order_number'],
print_data['card_number'],
               print_data['first_name'], print_data['last_name'], print_data['phone'],
print_data['order_date'],
               print_data['complaint']))
            print_thread.start()

            return jsonify({'success': True})
         except Exception as e:
            print("Помилка:", e)
            logging.error(f'Помилка print check: {e}')
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'})


      @app.route('/issue_card', methods=['POST'])
      @login_required
      def issue_card():
         try:
            order_number = request.form.get('order_number')
            ip_address = request.remote_addr
            user_name = current_user.username
            time_issue = time.strftime("%Y-%m-%d %H:%M")
            order_data = Postgresql.search_order(order_number)
            if order_data:
```

```
                card_number = order_data['card_number']
                order_number = order_data['order_number']
                success = Postgresql.issue_order_status(card_number, user_name, time_issue,
order_number)

                if success:
                    return jsonify({'success': True})
                else:
                    return jsonify({'success': False, 'message': 'Не вдалося оновити статус.'})
            else:
                return jsonify({'success': False, 'message': 'Замовлення не знайдено.'})
        except Exception as e:
            print("Помилка:", e)
            logging.error(f'Помилка issue_card: {e}')
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500
    @app.route('/repair', methods=['GET', 'POST'])
    @login_required
    @role_required('admin', 'expert')
    def home_repair():
        count_block_list = Postgresql.get_count_block_list()
        return render_template('repair.html', username=current_user.username,
role=current_user.role, count_block_list=count_block_list)


    @app.route('/analis', methods=['GET', 'POST'])
    @login_required
    @role_required('admin', 'expert', 'operator')
    def index():
        try:
            today = time.strftime('%Y-%m-%d')
            criteria = {'date_start': today, 'date_end': today}
            if request.method == 'POST':
                criteria['date_start'] = request.form.get('date_start')
                criteria['date_end'] = request.form.get('date_end')
                criteria['status'] = request.form.get('status')
                criteria['ip'] = request.form.get('ip')
                criteria['card_number'] = request.form.get('card_number')
                criteria['repair_date_start'] = request.form.get('repair_date_start')
                criteria['repair_date_end'] = request.form.get('repair_date_end')
                criteria['issue_date_start'] = request.form.get('issue_date_start')
                criteria['issue_date_end'] = request.form.get('issue_date_end')
                criteria['block-deblock-search'] = request.form.get('block-deblock-search')
                criteria['complaint'] = request.form.get('complaint')

            records = Postgresql.get_orders(criteria)
            count_block_list = Postgresql.get_count_block_list()

            return render_template('table.html', records=records, criteria=criteria,
username=current_user.username, role=current_user.role, count_block_list=count_block_list)
        except Exception as e:
            print("Помилка:", e)
            logging.error(f'Помилка analis: {e}')
```

```
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500


        @app.route('/export', methods=['GET'])
        @login_required
        @role_required('admin', 'expert')
        def export():
            try:
                criteria = {
                    'date_start': request.args.get('date_start'),
                    'date_end': request.args.get('date_end'),
                    'status': request.args.get('status'),
                    'ip': request.args.get('ip'),
                    'card_number': request.args.get('card_number'),
                    'repair_date_start': request.args.get('repair_date_start'),
                    'repair_date_end': request.args.get('repair_date_end'),
                    'issue_date_start': request.args.get('issue_date_start'),
                    'issue_date_end': request.args.get('issue_date_end')
                }

                records = Postgresql.get_orders(criteria)

                # Create a string buffer
                si = StringIO()
                cw = csv.writer(si)
                cw.writerow([
                    "Номер", "Прізвище", "Ім'я", "Телефон", "Скарга", "Дата заявки", "Статус",
                    "Номер замовлення", "IP", "Примітки", "Час коли взято в роботу", "Час коли
виконано",
                    "IP видачі", "Час видачі", "Діагноз", "Вирішення", "Серія", "UID", "Баланс"
                ])

                # Write data
                for record in records:
                    cw.writerow(record)

                output = make_response(si.getvalue())
                output.headers["Content-Disposition"] = "attachment; filename=table_export.csv"
                output.headers["Content-type"] = "text/csv"
                return output

            except Exception as e:
                print("Помилка:", e)
                logging.error(f'Помилка EXPORT: {e}')
                return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500

        @app.route('/inventory', methods=['GET', 'POST'])
        @login_required
        def inventory():
            try:
                if request.method == 'POST':
                    card_number = request.form.get('card_number')
```

```
                ip_address = request.remote_addr
                time_sent = time.strftime("%Y-%m-%d %H:%M:%S")

                Postgresql.set_inventory(card_number)

                return render_template('inventory.html', username=current_user.username,
role=current_user.role)

            else:
                return render_template('inventory.html', username=current_user.username,
role=current_user.role)

        except Exception as e:
            print("Помилка:", e)
            logging.error(f'Помилка inventory: {e}')
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500

    @app.route('/inventorization', methods=['GET', 'POST'])
    def inventorization():

        try:
            if request.method == 'POST':
                card_number = request.form.get('card_number')
                ip_address = request.remote_addr
                time_sent = time.strftime("%Y-%m-%d %H:%M:%S")
                Postgresql.set_inventorization(card_number, ip_address, time_sent)
                inventory_cards = Postgresql.get_inventory_cards()  # Функція для вибірки
записів
                return render_template('inventorization.html', inventory_cards=inventory_cards)

            else:
                inventory_cards = Postgresql.get_inventory_cards()  # Функція для вибірки
записів
                print(inventory_cards)
                return render_template('inventorization.html', inventory_cards=inventory_cards)

        except Exception as e:
            print("Помилка:", e)
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 50

    @app.route('/analytics', methods=['GET', 'POST'])
    @login_required
    @role_required('call_center', 'admin')
    def analytics():
        try:
            username = current_user.username
            statistic = Admin_panel.counter()

            return render_template('analytics.html', username=username, statistic=statistic)

        except Exception as e:
            print("Помилка:", e)
```

```
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500

        @app.route('/status', methods=['GET', 'POST'])
        @login_required
        def status():
            return render_template('status.html', username=current_user.username,
role=current_user.role)

        @app.route('/block_deblock', methods=['GET', 'POST'])
        @login_required
        def blockdeblock():
            try:
                if request.method == 'POST':
                    card_number = request.form.get('card_number')
                    if not card_number:
                        return jsonify({'error': 'Введіть номер карти'})

                    action = request.form.get('action')
                    reason = request.form.get('reason')
                    selected_reason = request.form.get('reason_select')
                    time_add = time.strftime("%Y-%m-%d %H:%M:%S")
                    user = current_user.username

                    success = Postgresql.block_deblock(card_number, action, reason, time_add, user,
selected_reason)

                    if success:
                        flash(f'Картка {card_number} буде  {"заблокована" if action == "Блок" else
"розблокована"}.',
                              'success')
                        socketio.emit('new_card', {
                            'message': f'Картку {card_number} потрібно {"заблокувати" if action ==
"Блок" else "розблокувати"}.'})
                    else:
                        flash('Не вдалося виконати операцію. Спробуйте ще раз.', 'error')

                    return redirect(url_for('blockdeblock'))

                reason_select = Postgresql.get_reason_block_deblock()
                return render_template('block_deblock.html', username=current_user.username,
reasons=reason_select, role=current_user.role)
            except Exception as e:
                print("Помилка:", e)
                flash('Сталася внутрішня помилка сервера.', 'error')
                return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500

        @app.route('/block_list', methods=['GET', 'POST'])
        @login_required
        def block_list():
            try:
                card_number = request.form.get('card_number')
                status = request.form.get('status')
```

```
            action = request.form.get('action')
            count_block_list = Postgresql.get_count_block_list()

            block_list = []

            if action == 'open':
                block_list = Postgresql.get_block_list()
            elif action == 'all':
                block_list = Postgresql.get_all_cards()
            else:
                if card_number and status:
                    block_list = Postgresql.get_block_list_by_card_and_status(card_number,
status)
                elif card_number:
                    block_list = Postgresql.get_block_list_card_number(card_number)
                elif status:
                    block_list = Postgresql.get_block_list_status(status)
                else:
                    block_list = Postgresql.get_block_list()
            return render_template('block_list.html', records=block_list,
username=current_user.username, role=current_user.role, count_block_list=count_block_list )
        except Exception as e:
            print("Помилка:", e)
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500

    @app.route('/execute_action', methods=['POST'])
    @login_required
    def execute_action():
        try:
            card_number = request.form.get('card_number')
            approve_reason = request.form.get('approve_reason')
            time_add = request.form.get('time_add')
            Postgresql.update_status(card_number, 'executed', approve_reason, time_add)

            return redirect(url_for('block_list'))
        except Exception as e:
            print("Помилка:", e)
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500

    @app.route('/cancel_action', methods=['POST'])
    @login_required
    def cancel_action():
        try:
            card_number = request.form.get('card_number')
            cancel_reason = request.form.get('cancel_reason')
            time_add = request.form.get('time_add')
            action = request.form.get('action')
            Postgresql.update_status(card_number, 'canceled', cancel_reason, time_add)
            socketio.emit('new_card', {
                'message': f'Картку {card_number} не можливо {"заблокувати" if action ==
"Блок" else "розблокувати"}. по причині: {cancel_reason}'})
            return redirect(url_for('block_list'))
```

```
        except Exception as e:
            print("Помилка:", e)
            return jsonify({'error': 'Сталася внутрішня помилка сервера.'}), 500

    @app.route('/get_more_records', methods=['GET'])
    def get_more_records():
        try:
            offset = int(request.args.get('offset', 0))
            limit = int(request.args.get('limit', 15))
            records = Postgresql.get_records_from_db(offset, limit)  # Тут ваша функція для
отримання записів

            return jsonify(records)
        except Exception as e:
            return jsonify({'error': f'Помилка бази даних: {str(e)}'}), 500


    @app.route('/search_card')
    def search_card():
        try:
            card_number = request.args.get('card_number')
            if not card_number:
                try:
                    offset = 0
                    limit = 15
                    records = Postgresql.get_records_from_db(offset, limit)
                    return jsonify(records)
                except Exception as e:
                    return jsonify({'error': f'Помилка бази даних: {str(e)}'}), 500
            try:
                cards = Postgresql.get_block_list_card_number(card_number)
            except Exception as e:
                return jsonify({'error': f'Помилка бази даних: {str(e)}'}), 500
            filtered_records = [record for record in cards if str(card_number) in str(record[0])]

            return jsonify(filtered_records)
        except Exception as e:
            return jsonify({'error': f'Помилка: {str(e)}'}), 500

    if __name__ == '__main__':
        app.run(host='0.0.0.0', port=config['DEFAULT']['PORT'], debug=True)
        socketio.run(app)
```

Postgresql.py

```
import psycopg2
from configparser import ConfigParser
import re
import logging
from datetime import datetime

config = ConfigParser()
config.read('config.ini')
conn = psycopg2.connect(dbname=config['DEFAULT']['BD_NAME'],
                        user=config['DEFAULT']['BD_USER'],
                        password=config['DEFAULT']['BD_PASS'],
                        host=config['DEFAULT']['BD_HOST'],
                        port=config['DEFAULT']['BD_PORT'])


def search_card(card_number):
    try:
        cur = conn.cursor()

            cur.execute('''
        SELECT *,
            CASE
                WHEN "ORDER_STATUS" = 141 THEN 'Очікує блокування'
                WHEN "ORDER_STATUS" = 142 THEN 'Заблокована'
                WHEN "ORDER_STATUS" = 145 THEN 'Очікує розблокування'
                WHEN "ORDER_STATUS" = 10 THEN 'Видана'
                WHEN "ORDER_STATUS" = 11 THEN 'Знищена'
                ELSE 'Невідомий статус'
            END AS "StatusText"
        FROM "Cards" WHERE "TN" = %s
        ''', (card_number,))

                rows = cur.fetchall()

        for row in rows:
            if row:
                print('Знайдено в БД -',row)
                logging.info('Знайдено в БД -%s',row)
                return row

            else:
                return None
        cur.close()

    except Exception as e:
        print(e)

    def add_orders(tn, last_name, name, phone_number, complaint, time, status,
order_number, user_ip, notes):
        try:
```

```
            cur = conn.cursor()
                  cur.execute('INSERT INTO "CardFlow_orders" ("TN", '
                     '"LAST_NAME", '
                     '"NAME", '
                     '"PERSON_MOBILE_PHONE",'
                     '"ORDER_PROBLEM", '
                     '"TIME_ADD", '
                     '"STATUS",'
                     '"NUMBER_ORDER", '
                     '"IP", '
                     '"COMMIT") VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)',
                     (tn,
                      last_name,
                      name,
                      phone_number,
                      complaint,
                      time,
                      status,
                      order_number,
                      user_ip,
                      notes
                      ))

                     conn.commit()

            print("Замовлення успішно додано до бази даних!")
            logging.info("Замовлення успішно додано до бази даних!")
         except Exception as e:
            print(f"An error occurred: {e}")
            conn.rollback()
         finally:
            cur.close()


      def standardize_phone_number(phone_number):
         try:
            digits = re.sub(r'\D', '', phone_number)
                  if digits.startswith('380'):
               standardized = f'+{digits[:3]} ({digits[3:5]}) {digits[5:8]}-{digits[8:10]}-
{digits[10:12]}'
            else:
               return None
            return standardized
         except Exception as e:
            print(e)
            return None

      def search_order(order_number):
         try:
            cur = conn.cursor()
            if order_number.startswith('*'):
               cur.execute('SELECT * FROM "CardFlow_orders" WHERE
```

```
"NUMBER_ORDER" = %s ORDER BY "TIME_ADD" DESC LIMIT 1', (order_number,))
            elif order_number.startswith('+380'):
                phone_number = standardize_phone_number(order_number)
                cur.execute(
                    'SELECT * FROM "CardFlow_orders" WHERE
"PERSON_MOBILE_PHONE" = %s ORDER BY "TIME_ADD" DESC LIMIT 1',
                    (phone_number,))
                if cur.rowcount == 0:
                    cur.execute('SELECT * FROM "CardFlow_orders" WHERE
"PERSON_MOBILE_PHONE" = %s ORDER BY "TIME_ADD" DESC LIMIT 1',
                                (order_number[3:],))
                    if cur.rowcount == 0:
                        cur.execute(
                            'SELECT * FROM "CardFlow_orders" WHERE
"PERSON_MOBILE_PHONE" = %s ORDER BY "TIME_ADD" DESC LIMIT 1',
                            (order_number,))
            else:
                cur.execute('SELECT * FROM "CardFlow_orders" WHERE "TN" = %s ORDER
BY "TIME_ADD" DESC LIMIT 1', (order_number,))

            result = cur.fetchone()

            if result:
                last_order_data = {
                    "status": result[6],
                    "card_number": result[0],
                    "first_name": result[2],
                    "last_name": result[1],
                    "phone": result[3],
                    "complaint": result[4],
                    "order_date": result[5],
                    "order_number": result[7],
                    "notes": result[9],
                    "repair_date": result[11],
                    # Додайте інші поля за потребою
                }
                return last_order_data
            else:
                return None

        except Exception as e:
            print(e)
            conn.rollback()
            return None
        finally:
            cur.close()

    def get_order_number():
        try:
            cur = conn.cursor()
            cur.execute('SELECT MAX("NUMBER_ORDER") FROM "CardFlow_orders"')
            max_number = cur.fetchone()[0]  # отримуємо перший результат з кортежу
```

```
            if max_number:
                    max_number = int(max_number[1:]) + 1
            else:
                     max_number = 1

            cur.close()
            return max_number
        except Exception as e:
            print(e)
            return None

    def issue_order_status(card_number, ip_address, time_issue, order_number):
        try:
            cur = conn.cursor()

            cur.execute('''
                UPDATE "CardFlow_orders"
                SET "STATUS" = 'Видана', "IP_ISSUE" = %s, "TIME_ISSUE" = %s
                WHERE "TN" = %s AND "NUMBER_ORDER" = %s
            ''', (ip_address, time_issue, card_number, order_number))

            conn.commit()

            print(f'Картка {card_number} видана о {time_issue} на IP {ip_address}')
            logging.info(f'Картка {card_number} видана о {time_issue} на IP {ip_address}')
            return True
        except Exception as e:
            print(f"An error occurred: {e}")
            conn.rollback()  # Відкат змін у разі помилки
            return False
        finally:
            if conn:
                cur.close()


    def get_dates(tn):
        try:
            cur = conn.cursor()
            cur.execute('SELECT "TIME_ADD" FROM "CardFlow_orders" WHERE "TN" =
%s ORDER BY "TIME_ADD" DESC', (tn,))
            dates = cur.fetchall()
            dates_list = [date[0] for date in dates]
            return dates_list
        except Exception as e:
            print(e)
            conn.rollback()
            return None

        finally:
            cur.close()
```

```
        def get_history(tn, time_add):
            try:
                cur = conn.cursor()
                cur.execute('SELECT * FROM "CardFlow_orders" WHERE "TN" = %s AND
"TIME_ADD" = %s', (tn, time_add))
                result = cur.fetchone()
                if result[16] is None:
                    series = get_tsot_series(tn)
                else:
                    series = result[16]

                uid = get_tsot_UID(tn)

                order_data = {
                    "diagnosis": result[14],
                    "complite_com": result[15],
                    "series": series,
                    "uid": result[17],
                    "balance": result[18],
                    "uid_tsot": uid,
                }
                return order_data

            except Exception as e:
                print(e)
                conn.rollback()
                return None
            finally:
                cur.close()

        def set_history(card_number, diagnosis, complite_com, series, uid, balance, notes,
time_add, time_end):
            try:
                cur = conn.cursor()

                cur.execute('''
                    UPDATE "CardFlow_orders"
                    SET "STATUS" = 'Відремонтована',  "TIME_END" = %s, "DIAGNOSIS" =
%s, "COMPLITE_COM" = %s, "SERIES" = %s, "UID" = %s, "BALANCE" = %s, "COMMIT"
= %s
                    WHERE "TN" = %s AND "TIME_ADD" = %s
                ''', (time_end, diagnosis, complite_com, series, uid, balance, notes, card_number,
time_add))

                conn.commit()
                return True
            except Exception as e:
                print(e)
                conn.rollback()
                return False
            finally:
```

```
            cur.close()


        def card_to_work(card_number, order_date, notes):
            try:
                cur = conn.cursor()

                cur.execute('''
                    UPDATE "CardFlow_orders"
                    SET "STATUS" = 'В роботі' , "COMMIT" = %s
                    WHERE "TN" = %s AND "TIME_ADD" = %s
                ''', (notes, card_number, order_date))

                conn.commit()
                return True
            except Exception as e:
                print(e)
                conn.rollback()
                return False
            finally:
                cur.close()


        def get_tsot_series(card_number):
            try:
                cur = conn.cursor()
                cur.execute('SELECT COUNT(*) FROM "Tsot_series" WHERE "TN" = %s',
(card_number,))
                count = cur.fetchone()[0]
                if count == 3:
                    cur.execute('SELECT "SERIES" FROM "Tsot_series" WHERE "TN" = %s
ORDER BY "SERIES" ASC LIMIT 1', (card_number,))
                    result = cur.fetchone()
                    if result:
                        return result[0]
                    else:
                        return None
            except Exception as e:
                print(e)
                conn.rollback()
                return None
            finally:
                cur.close()


        def get_tsot_UID(card_number):
            try:
                cur = conn.cursor()
                cur.execute('SELECT COUNT(*) FROM "Tsot_series" WHERE "TN" = %s',
(card_number,))
                count = cur.fetchone()[0]
                if count == 3:
```

```
                cur.execute('SELECT "CHIP" FROM "Tsot_series" WHERE "TN" = %s ORDER
BY "SERIES" ASC LIMIT 1', (card_number,))
                result = cur.fetchone()
                if result:
                    return result[0][:14]
                else:
                    return None
            except Exception as e:
                print(e)
                conn.rollback()
                return None
            finally:
                cur.close()


        def get_orders(criteria):
            records = []
            try:
                cur = conn.cursor()

                query = 'SELECT * FROM "CardFlow_orders" WHERE 1=1'
                params = []

                if criteria.get('date_start'):
                    query += ' AND "TIME_ADD"::date >= %s'
                    params.append(criteria['date_start'])
                if criteria.get('date_end'):
                    query += ' AND "TIME_ADD"::date <= %s'
                    params.append(criteria['date_end'])
                if criteria.get('status'):
                    query += ' AND "STATUS" = %s'
                    params.append(criteria['status'])
                if criteria.get('ip'):
                    query += ' AND "IP" = %s'
                    params.append(criteria['ip'])
                if criteria.get('card_number'):
                    query += ' AND "TN" = %s'
                    params.append(criteria['card_number'])
                if criteria.get('repair_date_start'):
                    query += ' AND "TIME_END"::date >= %s'
                    params.append(criteria['repair_date_start'])
                if criteria.get('repair_date_end'):
                    query += ' AND "TIME_END"::date <= %s'
                    params.append(criteria['repair_date_end'])
                if criteria.get('issue_date_start'):
                    query += ' AND "TIME_ISSUE"::date >= %s'
                    params.append(criteria['issue_date_start'])
                if criteria.get('issue_date_end'):
                    query += ' AND "TIME_ISSUE"::date <= %s'
                    params.append(criteria['issue_date_end'])
                if criteria.get('complaint'):
                    query += ' AND "ORDER_PROBLEM" ILIKE %s'
```

```
                params.append('%' + criteria['complaint'] + '%')

            query += ' ORDER BY "TIME_ADD" DESC'

            cur.execute(query, tuple(params))
            records = cur.fetchall()

            if criteria.get('block-deblock-search') and criteria.get('card_number'):
                query_block = '''
                    SELECT
                        "TN", NULL, NULL, NULL, "NEED_ACTION", "TIME_ADD",
"STATUS", NULL, "USER", "COMMIT", NULL, NULL, NULL, NULL, "REASON",
"COMMIT_ACTION",
                        NULL, NULL, NULL,   NULL
                    FROM "BLOCK_LIST"
                    WHERE "TN" = %s
                '''
                cur.execute(query_block, (criteria['card_number'],))
                block_records = cur.fetchall()
                records.extend(block_records)
                records.sort(key=lambda x: x[5] or datetime.min, reverse=True)
            return records
        except Exception as e:
            print("DB Error:", e)
            if 'conn' in globals() or 'conn' in locals():
                conn.rollback()
            return records
        finally:
            if 'cur' in locals() or 'cur' in globals():
                cur.close()
    #Функція для зміни статусу
    def set_inventory(card_number):
        try:
            cur = conn.cursor()

            card_data = search_order(card_number)

            cur.execute('''
                UPDATE "CardFlow_orders"
                SET "STATUS" = 'Готова до видачі'
                WHERE "TN" = %s AND "TIME_ADD" = %s
            ''', (card_data['card_number'], card_data['order_date']))
            conn.commit()

            print(f'Картка {card_number} переміщена на видачу')
            logging.info(f'Картка {card_number} переміщена на видачу')
            return True
        except Exception as e:
            print(f"An error occurred: {e}")
            conn.rollback()  # Відкат змін у разі помилки
            return False
        finally:
```

```
            if conn:
                cur.close()

        def set_inventorization(card_number, ip_address, time_sent):
            try:
                cur = conn.cursor()

                # Оновлення статусу замовлення
                cur.execute('" INSERT INTO "Inventory" ("TN", "TIME", "IP") VALUES (%s, %s,
%s)"', (card_number, time_sent, ip_address))
                conn.commit()

                print(f'Картка {card_number} проведена інвентаризація о {time_sent} на IP
{ip_address}')
                return True
            except Exception as e:
                print(f"An error occurred: {e}")
                conn.rollback()  # Відкат змін у разі помилки
                return False
            finally:
                if conn:
                    cur.close()


        def block_deblock(card_number, action, reason, time_add, user, selected_reason):
            try:
                # Відкрити курсор для взаємодії з базою даних
                cur = conn.cursor()

                cur.execute('"
                    INSERT INTO "BLOCK_LIST" ("TN", "NEED_ACTION", "COMMIT",
"TIME_ADD", "USER", "REASON")
                    VALUES (%s, %s, %s, %s, %s, %s)
                "', (card_number, action, reason, time_add, user, selected_reason))

                # Зафіксувати зміни у базі даних
                conn.commit()
                return True
            except Exception as e:
                # Вивести помилку, якщо виникла проблема
                print(f"An error occurred: {e}")
                # Відкотити зміни у разі помилки
                conn.rollback()
                return False
            finally:
                if conn:
                    cur.close()
        def get_block_list():
            try:
                cur = conn.cursor()
                cur.execute('SELECT * FROM "BLOCK_LIST" WHERE "STATUS" IS NULL
ORDER BY "TIME_ADD" DESC')
```

```
            records = cur.fetchall()
            return records
        except Exception as e:
            print(f"An error occurred: {e}")
            conn.rollback()
            return False
        finally:
            if conn:
                cur.close()
    def get_all_cards():
        try:
            cur = conn.cursor()
            cur.execute('SELECT * FROM "BLOCK_LIST"  ORDER BY "TIME_ADD"
DESC')
            records = cur.fetchall()
            return records
        except Exception as e:
            print(f"An error occurred: {e}")
            conn.rollback()
            return False
        finally:
            if conn:
                cur.close()
    def update_status(card_number, status, reason=None, time_add=None):
        try:
            cur = conn.cursor()
            if status == 'executed':
                # Оновлення статусу на "Виконано"
                cur.execute('''
                    UPDATE "BLOCK_LIST"
                    SET "STATUS" = %s, "COMMIT_ACTION" = %s
                    WHERE "TN" = %s AND "TIME_ADD" = %s
                ''', ('Виконано', reason, card_number, time_add))
            elif status == 'canceled':
                cur.execute('''
                    UPDATE "BLOCK_LIST"
                    SET "STATUS" = %s, "COMMIT_ACTION" = %s
                    WHERE "TN" = %s AND "TIME_ADD" = %s
                ''', ('Відмінено', reason, card_number, time_add))
            conn.commit()
            return True
        except Exception as e:
            print(f"An error occurred: {e}")
            conn.rollback()
            return False
        finally:
            if cur:
                cur.close()
```

Додаток Д. Ілюстративна частина

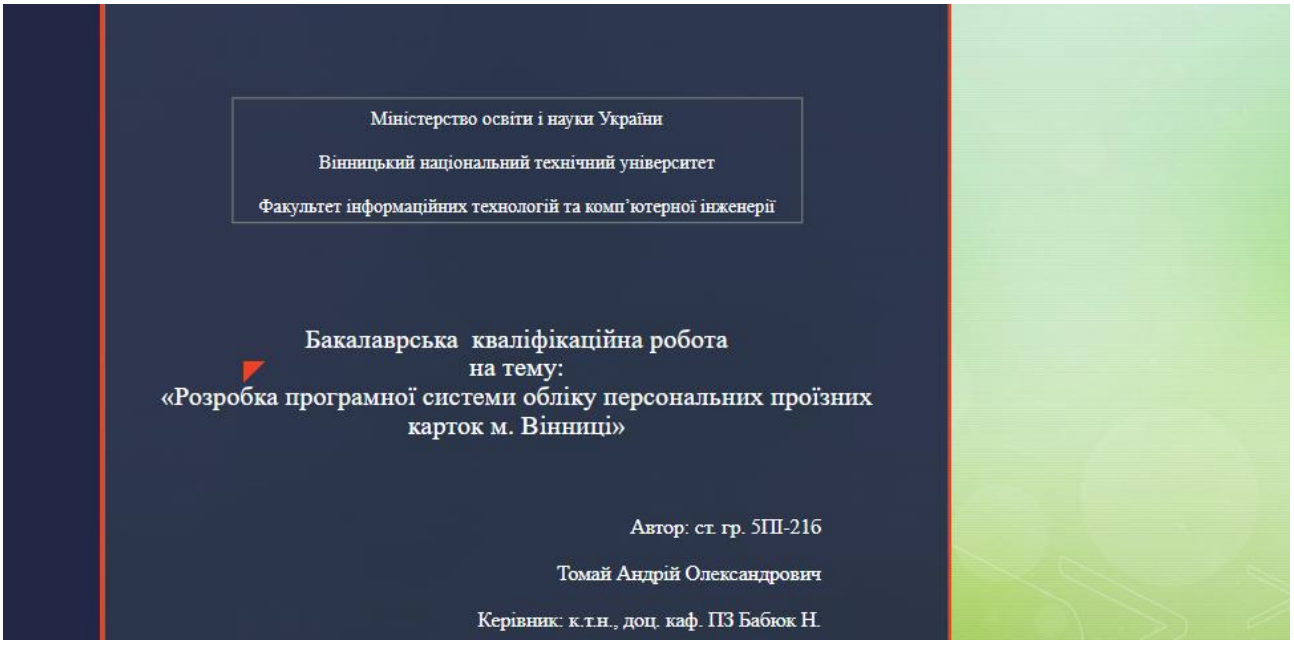

Рисунок Д.1 – Титульний слайд

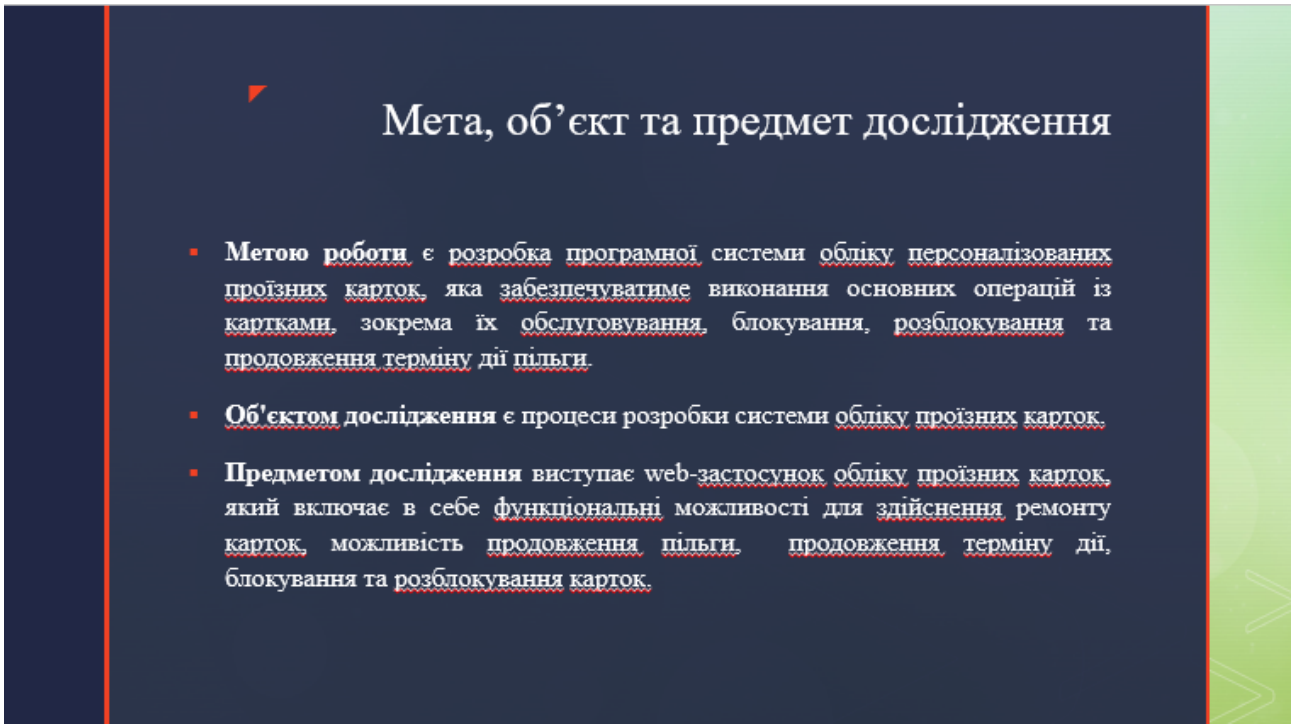

Рисунок Д.2 – Мета, об'єкт та предмет дослідження

Рисунок Д.3 – Актуальність розробки

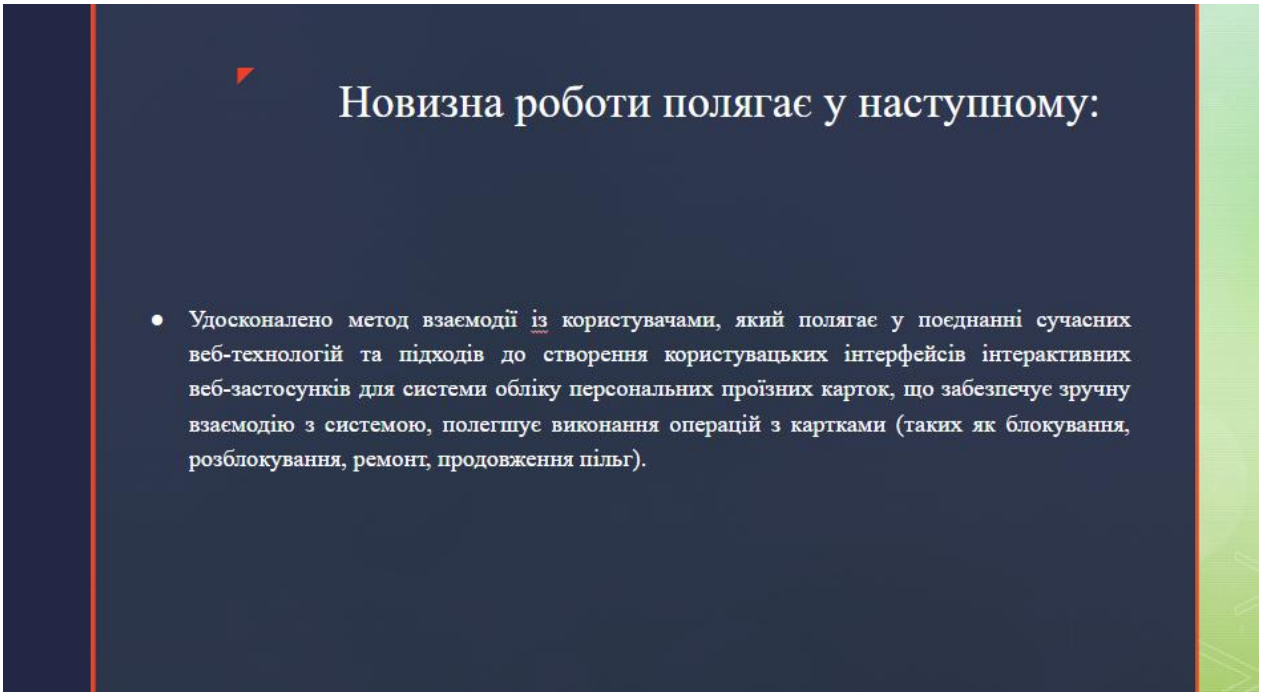

Рисунок Д.4 – Новизна роботи

Характеристика	RemOnline	Gincore	АСООП / ЛеоКарт	CardFlow
Облік ремонтів	+	+	–	+
Підтримка роботи з клієнтами (CRM)	+	+	–	+/–
Управління запасами (WMS)	+	+	–	–
Фінансовий облік	+/–	+	–	–
Повідомлення клієнтів	+	+	+	+
Хмарна архітектура	+	+	–	– (локальна мережа)
Кастомізація	–	+/–	–	+
Складність впровадження	+/–	–	+	Низька
Вартість	Висока	Висока	Середня	Низька
Адаптація під локальні задачі	–	–	–	+

Рисунок Д.5 – Порівняльний аналіз аналогів

Характеристики	Python + Flask	JavaScript (Node.js + Express)	PHP (Laravel)	Ruby (Ruby on Rails)
Простота використання	+	–	–	–
Гнучкість фреймворка	+	+	–	+
Ширина підтримки бібліотек	+	+	+	–
Швидкість розробки	+	–	+	–
Сумісність з іншими технологіями	+	+	+	+
Підтримка масштабованих додатків	+	+	+	+
Легкість у підтримці	+	–	–	–
Підтримка адаптивного дизайну	+	+	+	+

Рисунок Д.6 – Вибір засобів для реалізації застосунку

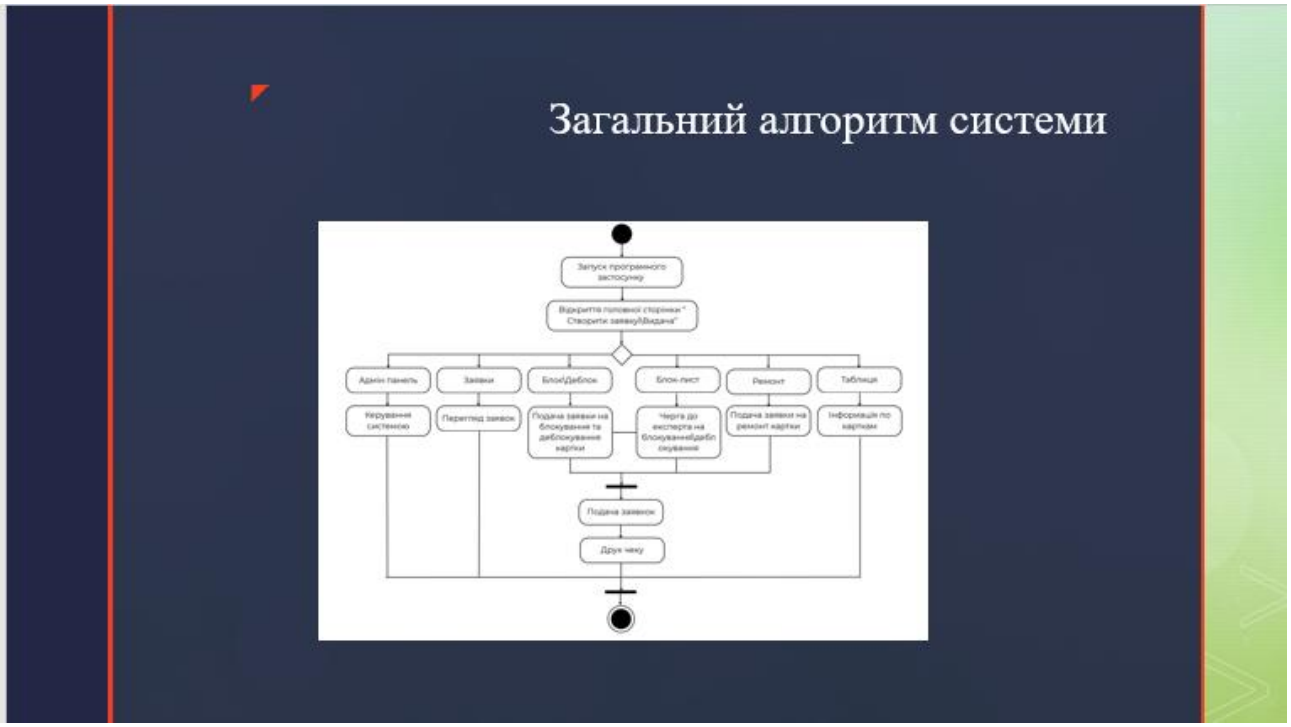

Рисунок Д.7–Загальний алгоритм системи

Рисунок Д.8 – Алгоритм прийому і видачі карток

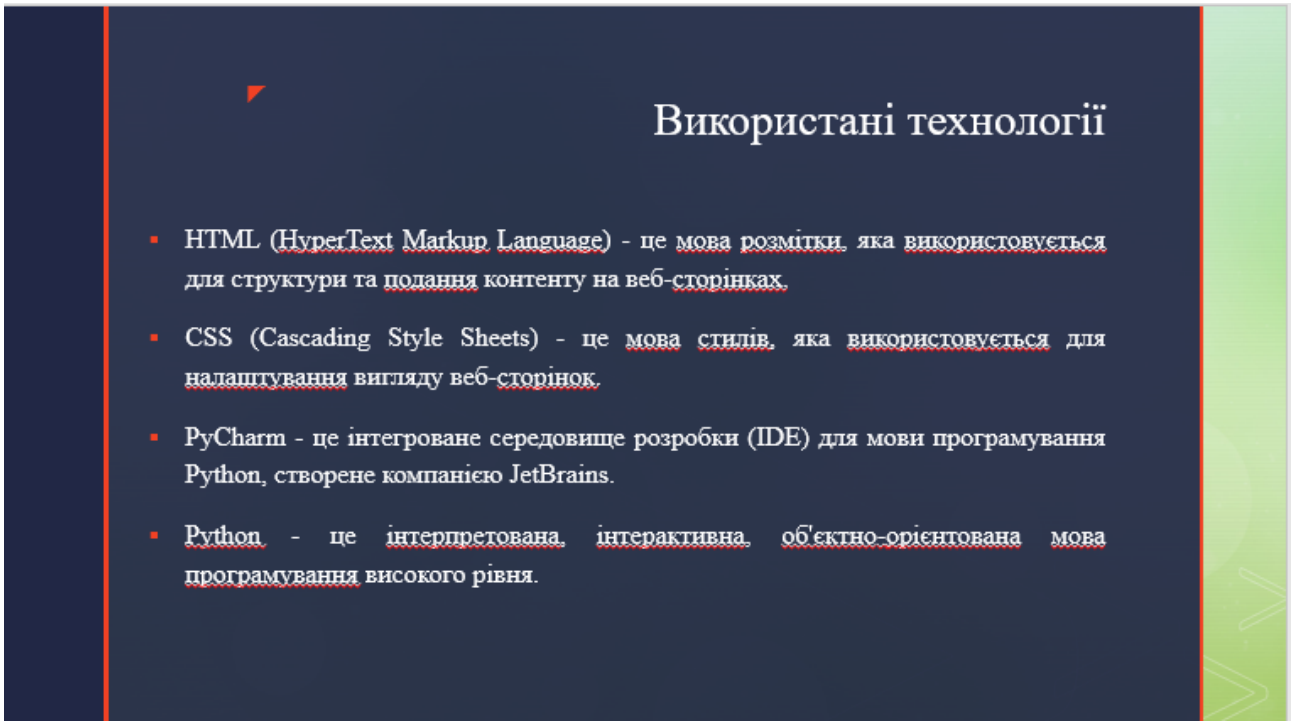

Рисунок Д.9 – Використані технології

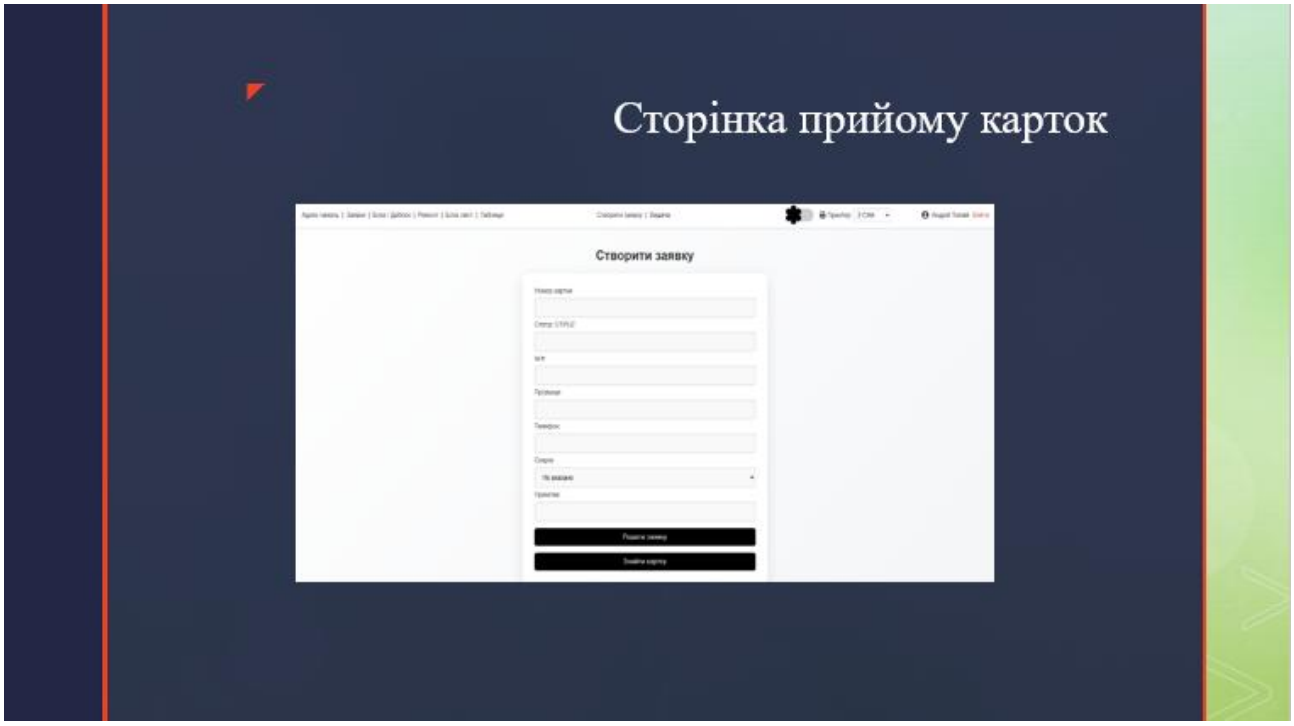

Рисунок Д.10 – Сторінка прийому карток

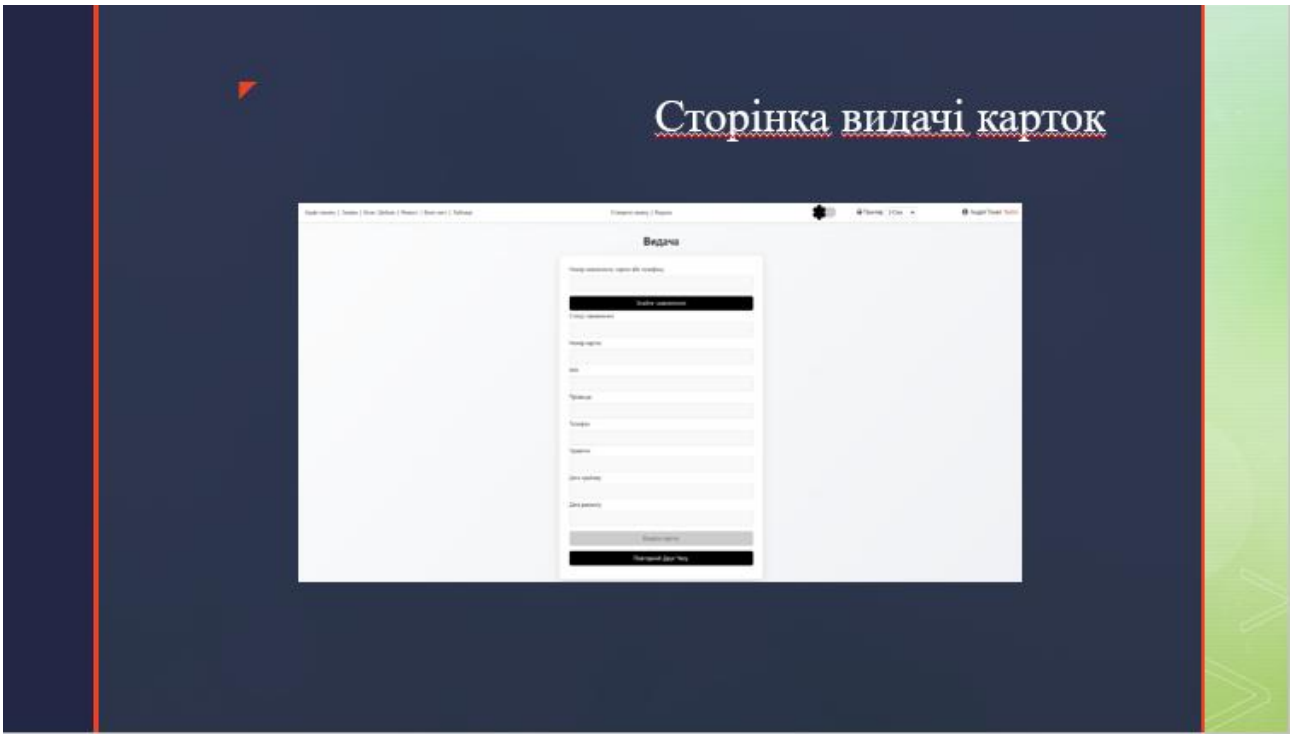

Рисунок Д.11 – Сторінка видачі карток

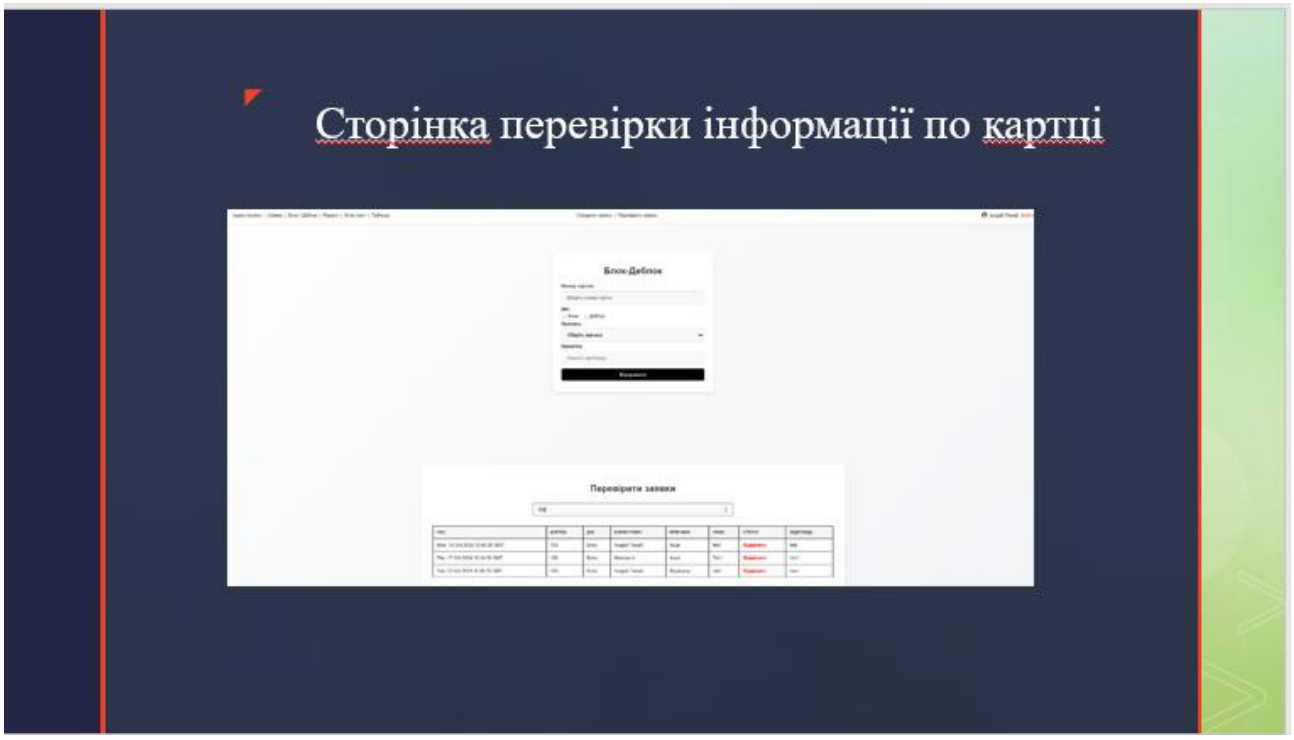

Рисунок Д.12 – Сторінка перевірки інформації по картці

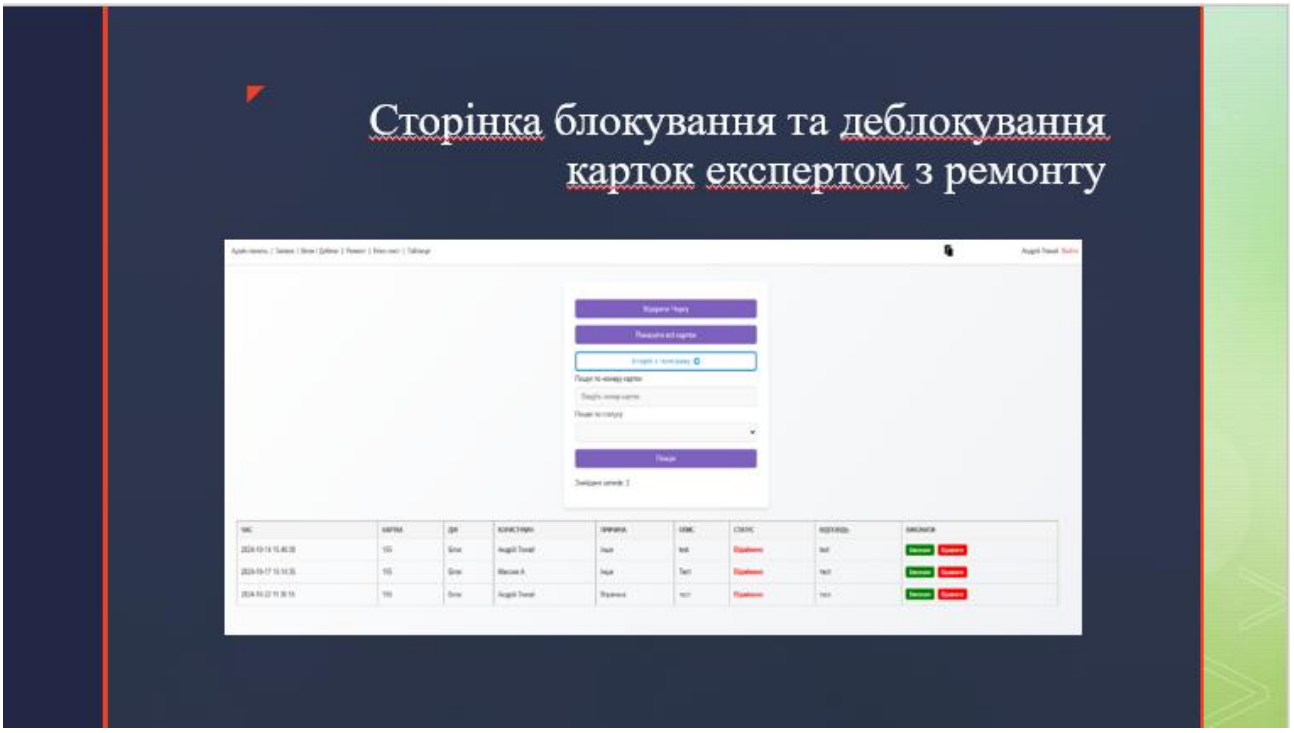

Рисунок Д.13 - Сторінка блокування та деблокування карток експертом з ремонту

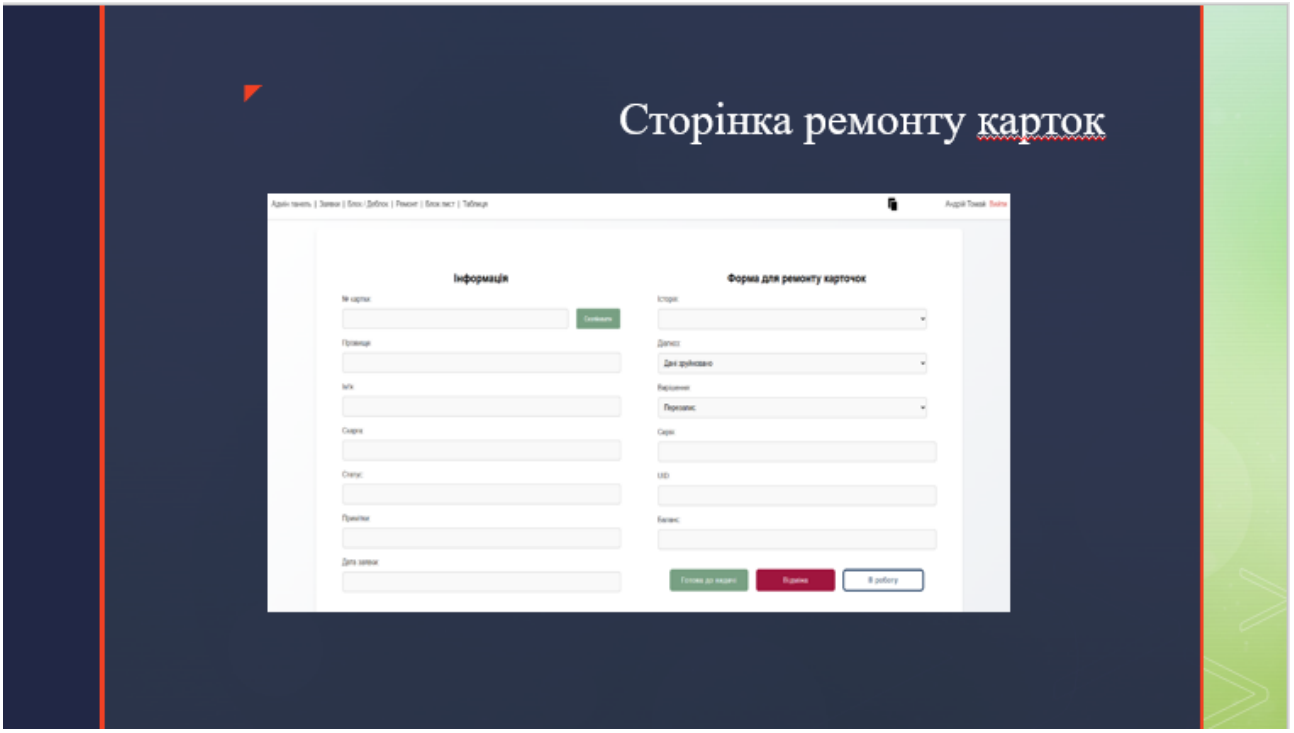

Рисунок Д.14 - Сторінка ремонту карток

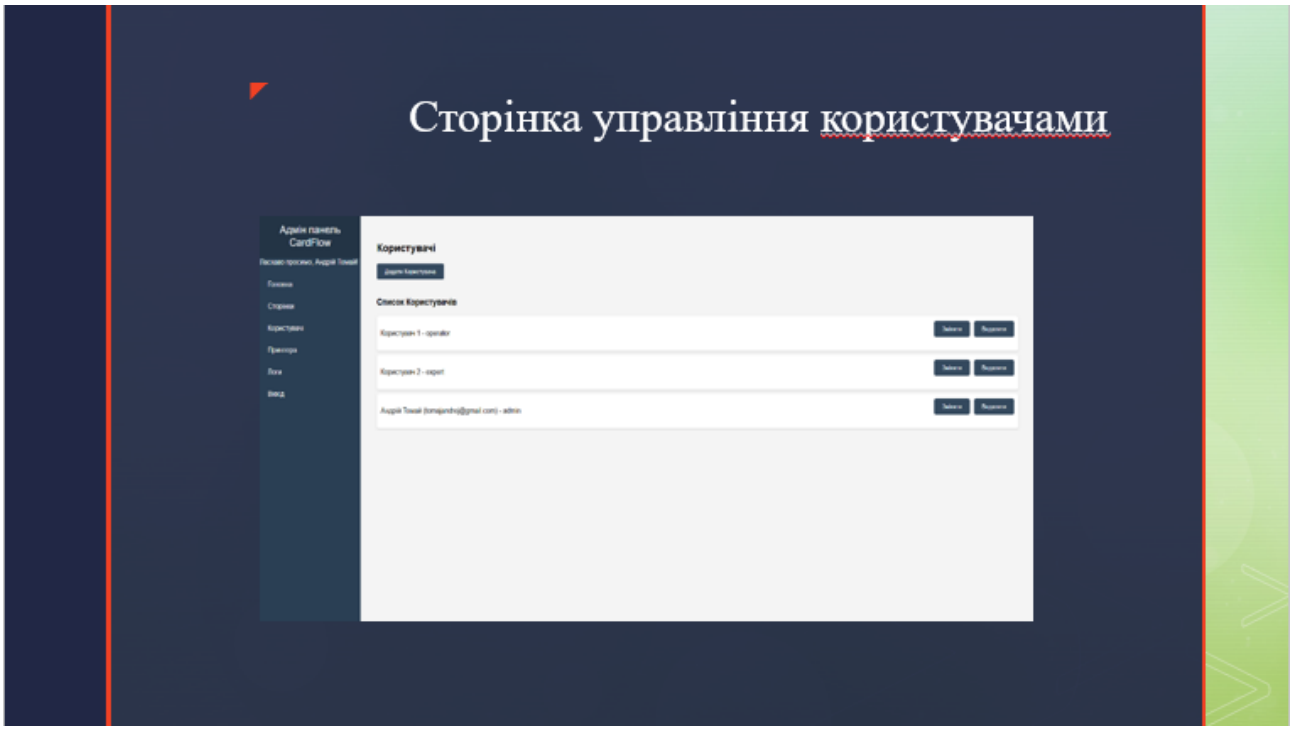

Рисунок Д.15 - Сторінка управління користувачами

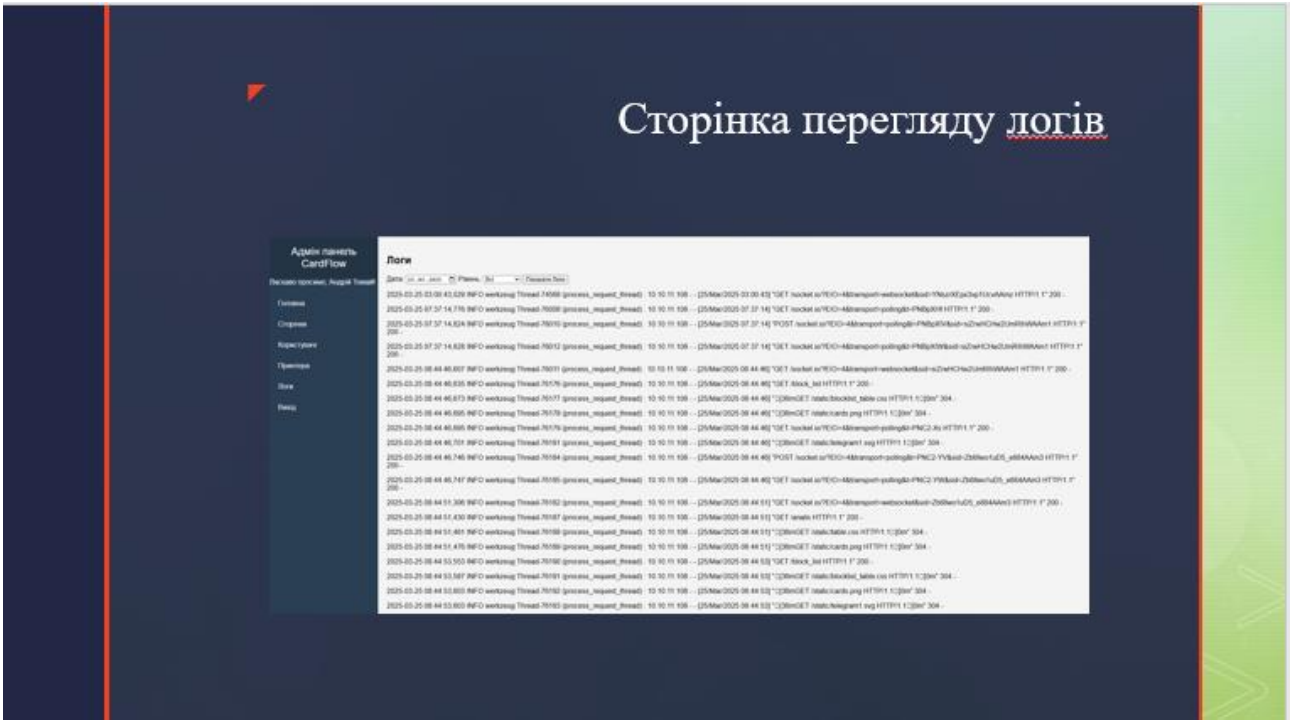

Рисунок Д.16 - Сторінка перегляду логів

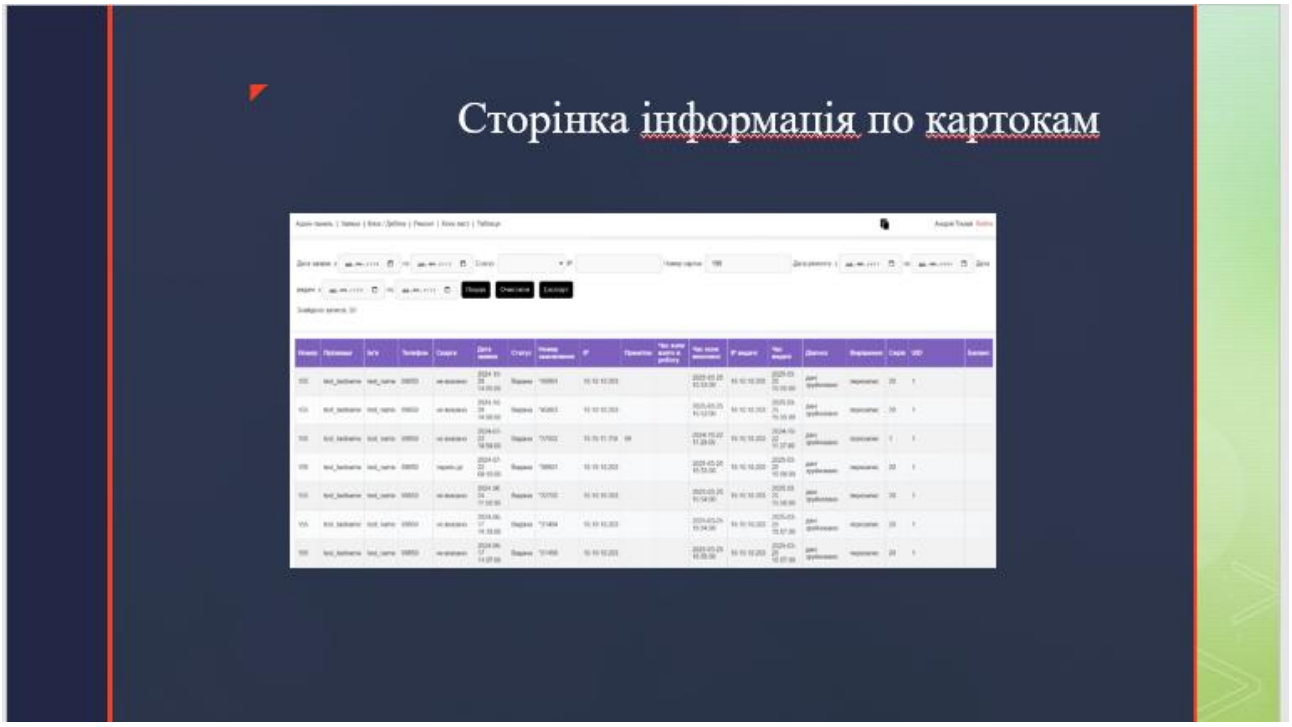

Рисунок Д.17 - Сторінка інформація по картокам

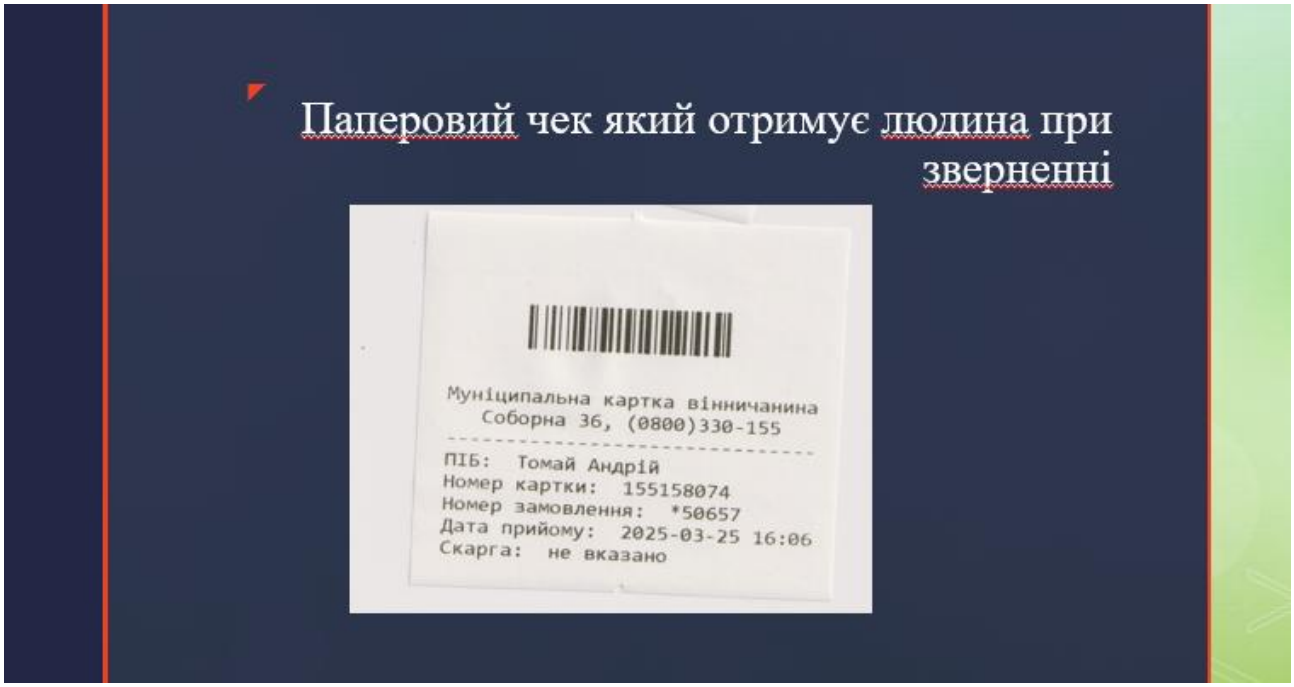

Рисунок Д.18 - Паперовий чек який отримує людина при зверненні

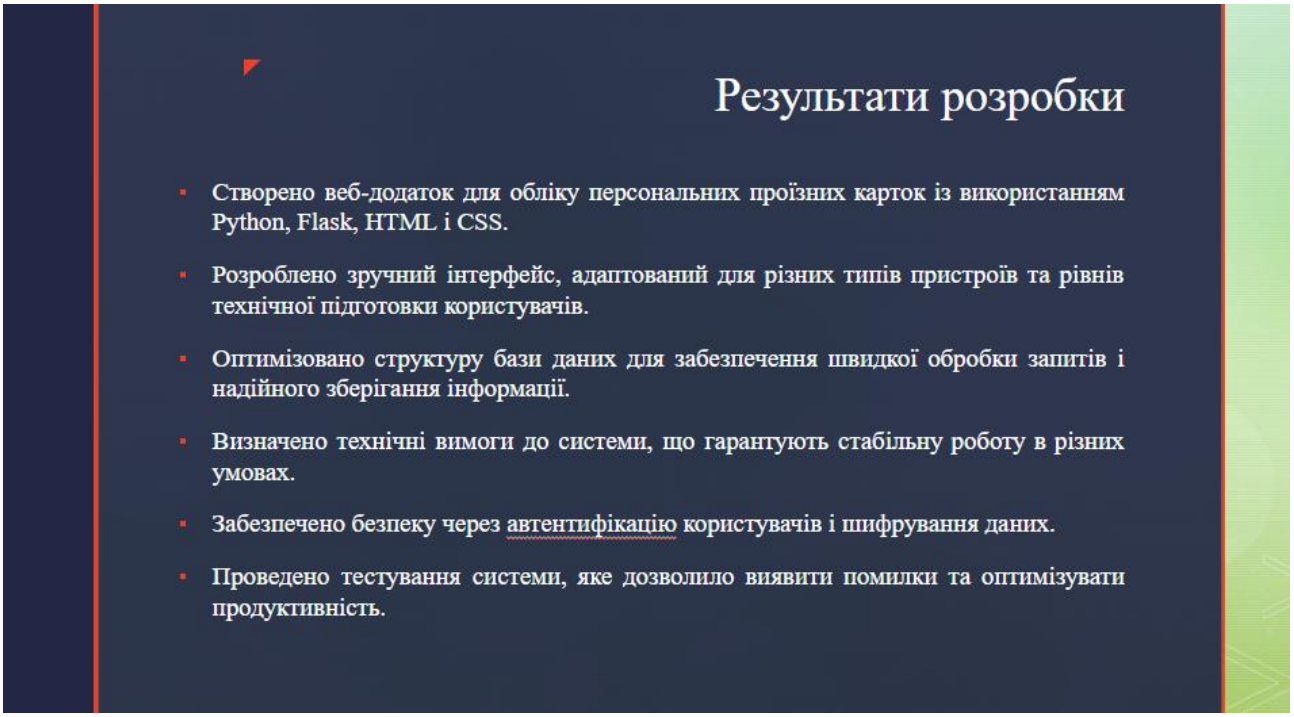

Рисунок Д.19 - Результати розробки

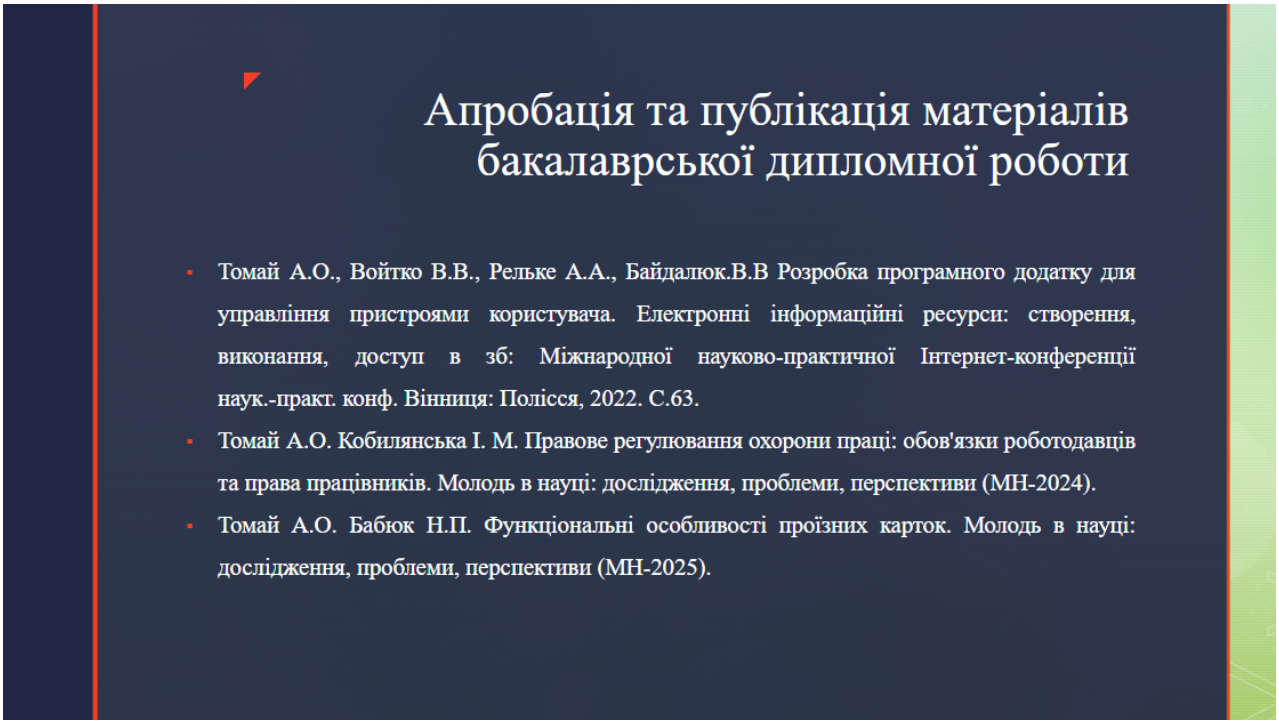

Рисунок Д.20 - Апробація та публікація матеріалів бакалаврської кваліфікаційної роботи

Рисунок Д.21 - Фінальний слайд