БАКАЛАВРСЬКА РОБОТА

РБ. ІП - 17.00.00.000 ПЗ

Група ІП-22-1

Мельничук Денис Іванович

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Мельничук Денис Іванович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Розроблення веб-орієнтованої інформаційної системи обліку ресурсів

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____Мельничук Д.І._____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____Процюк Василь Романович, к.т.н., доцент _____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.___
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу
Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2026 р.

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Мельничуку Денису Івановичу
(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Розроблення веб-орієнтованої інформаційної системи обліку ресурсів"

керівник проекту (роботи) Процюк Василь Романович, к.т.н., доцент _____

затверджені наказом вищого навчального закладу від **“** _21_ ” _квітня 2026 р. № 179/7_____

2. Строк подання студентом проекту (роботи) 10 червня 2026р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

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

1 Огляд проблеми обліку та управління ресурсами у веб-орієнтованих системах

2 Аналіз вимог користувачів і постановка задачі проєктування системи обліку ресурсів

3 Побудова архітектури, моделювання процесів та вибір технологій розробки

4 Документування програмної реалізації веб-орієнтованої системи обліку ресурсів

5 Тестування, впровадження, супровід та напрями розвитку системи

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

1 Діаграма варіантів використання системи (рис. 2.1, ст. 21)

2 Архітектура веб-орієнтованої системи обліку ресурсів (рис. 3.1, ст. 26)

3 Структура проєкту веб-орієнтованої системи обліку ресурсів (рис. 4.1, ст. 35)

4 Головна панель системи обліку ресурсів (рис. 4.3, ст. 39)

5 Каталог ресурсів із пошуком і фільтрацією (рис. 4.5, ст. 43)

6. Консультанти розділів проекту (роботи)

Розділ	*Консультант*	*Підпис, дата*	
		Завдання видав	*Завдання прийняв*

2. Дата видачі завдання **10 січня 2026 р.**

Керівник _____Процюк В.Р._____
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____Мельничук Д.І.___
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	*Назва етапів дипломного проекту (роботи)*	*Строк виконання етапів проекту*	*Примітка*
1	Визначення та обгрунтування теми роботи	15.02.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	03.03.2026	виконано
3	Побудова моделі або алгоритму власного рішення	25.03.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	10.04.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	10.06.2026	виконано

Студент-дипломник _____Мельничук Д.І._____
(підпис) (розшифровка підпису)

Керівник роботи _____Процюк В.Р._____
(підпис) (розшифровка підпису)

АНОТАЦІЯ

Бакалаврська робота містить 74 сторінок, 21 рисунків, 7 таблиць, список використаних джерел із 36 найменувань, 2 додатки.

Метою бакалаврської роботи є проєктування та реалізація веб-орієнтованої інформаційної системи обліку ресурсів ResourceFlow, що забезпечує ведення каталогу ресурсів, керування категоріями, роботу із заявками, рольовий доступ, статистику та збереження даних.

Об'єкт дослідження: процес обліку, контролю стану та використання ресурсів у веб-орієнтованій інформаційній системі.

Предмет дослідження: методи, програмні засоби та архітектурні рішення для створення веб-системи обліку ресурсів.

Результати дослідження: реалізовано систему ResourceFlow для обліку ресурсів, керування категоріями, створення заявок, перегляду статистики та збереження інформації.

У першому розділі розглянуто предметну область, проблеми обліку ресурсів та сучасні технології веб-систем.

У другому розділі визначено ролі користувачів, функціональні та нефункціональні вимоги і постановку задачі.

У третьому розділі спроєктовано архітектуру системи, бізнес-процеси та обґрунтовано вибір технологій.

У четвертому розділі описано реалізацію серверної частини, інтерфейсу, каталогу ресурсів, категорій, заявок і звітності.

У п'ятому розділі розглянуто тестування, впровадження, супровід, ризики та напрями розвитку системи.

Висновки: отримано працездатний прототип ResourceFlow, який підтримує авторизацію, рольовий доступ, CRUD-операції з ресурсами, заявки, статистику, історію змін, фото ресурсу та збереження даних.

КЛЮЧОВІ СЛОВА: ОБЛІК РЕСУРСІВ, RESOURCEFLOW, ВЕБ-СИСТЕМА, РОЛЬОВИЙ ДОСТУП, JAVASCRIPT, NODE.JS, JSON, CRUD, ЗАЯВКИ, СТАТИСТИКА.

ABSTRACT

The bachelor's thesis contains 74 pages, 21 figures, 7 tables, a list of 36 references, and 2 appendices.

The aim of the bachelor's thesis is to design and implement the ResourceFlow web-oriented information system for resource accounting, which provides resource catalog management, category management, request processing, role-based access, statistics, and data storage.

Object of research: the process of accounting, status control, and use of resources in a web-oriented information system.

Subject of research: methods, software tools, and architectural solutions for creating a web-based resource accounting system.

Research results: the ResourceFlow system for resource accounting, category management, request creation, statistics viewing, and information storage has been implemented.

The first chapter considers the subject area, resource accounting issues, and modern web system technologies.

The second chapter defines user roles, functional and non-functional requirements, and the task statement.

The third chapter presents the system architecture, business processes, and justification of the selected technologies.

The fourth chapter describes the implementation of the server side, interface, resource catalog, categories, requests, and reporting.

The fifth chapter considers testing, deployment, maintenance, risks, and directions for further system development.

Conclusions: a working ResourceFlow prototype has been obtained, supporting authorization, role-based access, CRUD operations with resources, requests, statistics, change history, resource photos, and data storage.

KEYWORDS: RESOURCE ACCOUNTING, RESOURCEFLOW, WEB SYSTEM, ROLE-BASED ACCESS, JAVASCRIPT, NODE.JS, JSON, CRUD, REQUESTS, STATISTICS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – прикладний програмний інтерфейс, через який клієнтська частина обмінюється даними із сервером.

CRUD – набір базових операцій створення, читання, редагування та видалення даних.

CSS – мова опису стилів веб-сторінки.

HTML – мова гіпертекстової розмітки для структурування сторінок.

HTTP – протокол передавання даних між браузером і сервером.

JSON – текстовий формат обміну та збереження структурованих даних.

JS – мова JavaScript, що використовується для логіки клієнтської частини.

UML – уніфікована мова моделювання програмних систем.

VPS – віртуальний приватний сервер для розміщення веб-застосунку.

ПЗ – програмне забезпечення.

СУБД – система управління базами даних.

ВСТУП

У наш час відбувається активна цифровізація різних сфер діяльності людини, що дає змогу спростити рутинні процеси, зменшити кількість помилок і підвищити швидкість обробки інформації. Особливо це стосується обліку ресурсів, оскільки техніка, обладнання, матеріали, меблі, інструменти, транспорт або інші об’єкти потребують постійного контролю, оновлення статусів і впорядкованого зберігання даних. Використання веб-орієнтованих інформаційних систем дозволяє замінити паперові журнали або окремі електронні таблиці єдиним середовищем для роботи з ресурсами.

Актуальність теми зумовлена потребою організацій у зручних і доступних інструментах для ведення обліку ресурсів. У багатьох випадках інформація про майно або матеріали зберігається розрізнено, що ускладнює пошук, контроль відповідальних осіб, визначення поточного стану ресурсу та формування звітності. Проблема стає особливо помітною тоді, коли один і той самий ресурс може видаватися різним користувачам, перебувати на ремонті, бути списаним або очікувати повернення. Тому створення універсальної веб-системи обліку ресурсів є актуальним практичним завданням для невеликих організацій, пунктів прокату, офісів і навчальних закладів.

Метою роботи є розробка веб-орієнтованої інформаційної системи обліку ресурсів ResourceFlow з інтуїтивним інтерфейсом, рольовим доступом, каталогом ресурсів, заявками, статистикою та збереженням даних. Система повинна забезпечувати зручну роботу з різними типами ресурсів і підтримувати основні дії користувача без потреби у спеціальній технічній підготовці.

Завданнями дослідження є аналіз предметної області, визначення ролей користувачів, формування функціональних і нефункціональних вимог, побудова архітектури системи, розробка клієнтської та серверної частини, реалізація обліку ресурсів, категорій, заявок, історії змін, статистики та перевірка працездатності системи. Окремим завданням є забезпечення адаптивності

інтерфейсу для роботи на різних пристроях і збереження даних після виходу користувача із системи.

Об’єктом дослідження є процеси обліку, контролю стану, використання та видачі ресурсів у веб-орієнтованій інформаційній системі. До цих процесів належать фіксація наявних ресурсів, оновлення їх статусів, призначення відповідальних осіб і контроль заявок на використання ресурсів.

Предметом дослідження виступає проєктування та реалізація веб-системи обліку ресурсів із використанням клієнтської частини, серверного API, рольового доступу та файлової бази даних. До предметної області також належить організація інтерфейсу користувача та структура збереження даних.

Методи дослідження включають аналіз предметної області, моделювання вимог користувачів, проєктування архітектури веб-застосунку, розробку з використанням HTML, CSS, JavaScript і Node.js, роботу з JSON-файлом як сховищем даних, а також функціональне тестування основних сценаріїв системи.

Наукова новизна дослідження полягає у поєднанні універсальної моделі обліку ресурсів із веб-орієнтованим інтерфейсом, рольовим доступом, заявками на використання ресурсів і можливістю адаптації системи під різні сфери діяльності без прив’язки до однієї конкретної галузі.

Практичне значення роботи полягає в створенні функціонального веб-застосунку ResourceFlow, який може використовуватися для обліку техніки, обладнання, матеріалів, меблів, інструментів, транспорту або інших ресурсів. Система дозволяє зберігати інформацію, змінювати статуси, створювати заявки, переглядати статистику та підтримувати актуальний стан ресурсів.

Бакалаврська робота містить 74 сторінок, 21 рисунків, 5 розділів, список використаних джерел із 36 найменувань, 2 додатки.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕОРЕТИЧНІ ОСНОВИ ПРОЄКТУВАННЯ СИСТЕМИ

1.1. Аналіз сучасного стану проблематики

У сучасних організаціях облік ресурсів є важливою частиною управління майном, матеріалами, обладнанням та іншими об’єктами, які використовуються у щоденній діяльності. До таких ресурсів можуть належати комп’ютерна техніка, меблі, інструменти, транспорт, витратні матеріали, спортивний інвентар або програмні ліцензії. Незалежно від сфери використання, відповідальним особам потрібно знати, які ресурси є в наявності, де вони розміщені, у якому стані перебувають і хто відповідає за їх використання [1].

Проблема ускладнюється тоді, коли кількість ресурсів зростає, а облік продовжує вестися вручну або у звичайних таблицях. У такому випадку виникають помилки в записах, дублювання інформації, складність пошуку потрібного ресурсу та відсутність єдиної історії змін. Крім того, користувачам важко швидко зрозуміти, чи доступний певний ресурс, чи він уже використовується, перебуває на ремонті або списаний. Через це відповідальна особа витрачає більше часу на уточнення даних, а рішення щодо видачі або переміщення ресурсу приймаються повільніше.

Особливо помітною ця проблема стає у випадках, коли ресурси не просто зберігаються на складі, а постійно переміщуються, видаються різним користувачам або мають змінний технічний стан. Наприклад, обладнання може бути доступним, переданим у користування, тимчасово несправним або списаним. Якщо ці стани не фіксуються в єдиній системі, відповідальна особа змушена уточнювати інформацію вручну, що збільшує витрати часу та створює ризик помилок [3].

Додатковою проблемою є відсутність прозорого контролю відповідальності. Якщо інформація про ресурс зберігається в різних таблицях або паперових записах, складно швидко визначити, хто останнім змінював дані,

кому було передано ресурс і чому його статус змінився. У результаті облік стає менш надійним, а процес управління ресурсами залежить не від системи, а від уважності окремих працівників.

У таблиці 1.1 наведено порівняння основних підходів до ведення обліку ресурсів. Таке порівняння дозволяє визначити переваги й недоліки традиційних та сучасних способів організації обліку. Воно також показує, чому для ефективного контролю ресурсів доцільно використовувати веб-орієнтовану систему, яка поєднує централізоване збереження даних, швидкий пошук, статуси та розмежування доступу.

Таблиця 1.1

Порівняння підходів до обліку ресурсів

Підхід	Характеристика	Обмеження
Паперовий журнал	Простий спосіб фіксації ресурсів без технічної інфраструктури.	Складний пошук, відсутність статистики, високий ризик втрати або дублювання даних.
Електронна таблиця	Дає змогу швидко створити перелік ресурсів і виконувати базову фільтрацію.	Не забезпечує ролей доступу, заявок, історії змін і повноцінної роботи кількох користувачів.
Галузева система	Може містити розвинені функції для складу, оренди або підприємства.	Часто є складною, дорогою або прив’язаною до конкретної сфери діяльності.
Універсальна веб-система	Підтримує доступ через браузер, ролі, заявки, статуси, статистику та централізоване збереження даних.	Потребує проєктування структури даних, серверної частини та зручного інтерфейсу.

Аналіз показує, що паперові журнали та електронні таблиці можуть бути корисними лише на початковому етапі, коли кількість ресурсів є невеликою. Проте вони не забезпечують повноцінної взаємодії між користувачами, не підтримують автоматизовані заявки та не дають змоги швидко отримувати статистику. Саме тому актуальним є створення веб-орієнтованої системи, яка поєднує простоту використання з базовими можливостями інформаційної системи.

Ще одним недоліком неавтоматизованого обліку є складність контролю відповідальності. У паперовому журналі або таблиці можна записати ім’я

відповідального, але важко швидко простежити, хто і коли змінив статус ресурсу, створив заявку або відредагував запис. У результаті втрачається прозорість процесу, а під час виникнення спірних ситуацій складно встановити послідовність дій [2].

Для невеликих організацій важливо, щоб система обліку не була надмірно складною. Великі корпоративні рішення часто мають значну кількість модулів, потребують налаштування, навчання персоналу та фінансових витрат. Водночас прості таблиці вже не покривають потреби в рольовому доступі, історії змін, заявках і статистиці. Тому актуальним є проміжне рішення, яке залишається простим для користувача, але підтримує ключові процеси інформаційної системи.

Система ResourceFlow розглядається саме як такий варіант. Вона орієнтована на універсальний облік ресурсів і не обмежується однією галуззю. Її можна застосовувати для техніки, меблів, транспорту, інструментів, матеріалів або інших об’єктів, які потрібно реєструвати, контролювати та видавати за заявками. Це робить систему придатною для різних умов використання.

Таким чином, сучасний стан проблематики показує потребу в доступному веб-рішенні, яке забезпечує централізований каталог ресурсів, зрозумілі статуси, рольовий доступ, фіксацію заявок і базову аналітику. Саме ці потреби стали підставою для подальшого проєктування системи ResourceFlow [4].

1.2. Основні поняття та визначення предметної області

Предметною областю роботи є процес обліку ресурсів у веб-орієнтованій інформаційній системі. Під ресурсом у межах цієї роботи розуміється будь-який об’єкт, який може бути зареєстрований у системі, мати власні характеристики, статус, категорію та відповідальну особу.

Ресурсом може бути ноутбук, принтер, велосипед, інструмент, стіл, матеріал, інвентар або інший об’єкт, який потрібно контролювати. Такий підхід дозволяє використовувати систему не лише для однієї конкретної сфери, а для

різних організаційних потреб, де важливо зберігати актуальну інформацію про наявні ресурси.

Категорія ресурсу використовується для групування об’єктів за спільними ознаками. Наприклад, ресурси можуть бути поділені на техніку, транспорт, інструменти, матеріали або обладнання. Наявність категорій спрощує пошук, фільтрацію та аналіз даних, оскільки користувач може працювати не з усім переліком одразу, а з конкретною групою ресурсів [5].

Статус ресурсу відображає його поточний стан у системі. Для ResourceFlow доцільно використовувати такі статуси, як «доступний», «зайнятий», «на ремонті» та «списаний». Завдяки статусам користувач може швидко оцінити, чи можна використовувати ресурс, чи він тимчасово недоступний. Це особливо важливо для систем, пов’язаних із видачею або орендою ресурсів.

Заявка є записом, який фіксує намір користувача отримати або використати певний ресурс. Вона містить інформацію про ресурс, користувача, мету використання, дату створення та поточний статус. Заявки дозволяють упорядкувати процес видачі ресурсів і зробити його контрольованим для менеджера або адміністратора [6].

Окремим поняттям є роль користувача. У системі передбачено адміністратора, менеджера та звичайного користувача. Роль визначає доступні дії: адміністратор керує всією системою, менеджер працює з ресурсами та заявками, а користувач переважно переглядає доступні ресурси та створює заявки. Такий поділ забезпечує впорядкованість і зменшує ризик помилкових дій.

Важливим поняттям предметної області є також інвентарний код. Він використовується для однозначної ідентифікації ресурсу в системі. На відміну від назви, яка може повторюватися для кількох однакових об’єктів, код дозволяє точно визначити конкретну одиницю майна.

Місце зберігання є характеристикою, яка показує, де фізично розташований ресурс. Для універсальної системи це може бути кабінет, склад,

пункт прокату, майстерня, аудиторія або інша локація. Наявність такого поля допомагає швидко знайти ресурс і зменшує час на його пошук у реальному середовищі.

Відповідальна особа вказує користувача або працівника, який контролює ресурс чи тимчасово відповідає за його стан. Це поняття має практичне значення, оскільки облік ресурсів пов’язаний не лише з наявністю об’єктів, а й з відповідальністю за їх використання. У системі така інформація дозволяє швидко встановити, до кого звертатися у разі уточнення стану ресурсу.

Історія змін є елементом, який фіксує важливі дії в системі. До таких дій можна віднести додавання нового ресурсу, редагування його даних, зміну статусу, створення заявки або зміну її стану. Журнал подій підвищує прозорість роботи системи та дозволяє відстежувати, як змінювалися дані протягом часу.

Життєвий цикл ресурсу охоплює послідовність станів, через які проходить об’єкт у процесі використання. Спочатку ресурс додається до системи, після чого він може бути доступним для користувачів, переданим у користування, тимчасово відправленим на ремонт або списаним. Такий підхід дозволяє розглядати ресурс не як статичний запис у таблиці, а як об’єкт, стан якого змінюється залежно від реальних подій.

Картка ресурсу є структурованим набором даних про конкретний об’єкт. Вона може містити назву, категорію, інвентарний код, місце зберігання, відповідальну особу, статус, вартість і фото. Наявність картки ресурсу спрощує перегляд інформації, оскільки всі важливі відомості зібрані в одному місці. Це зручно для адміністратора, менеджера та користувача, який хоче швидко зрозуміти, чи підходить ресурс для його потреб.

Ще одним важливим поняттям є доступність ресурсу. Доступність означає можливість використання ресурсу в конкретний момент часу. Наприклад, ресурс може бути фізично наявним у системі, але недоступним через ремонт або вже створену заявку. Тому в інформаційній системі важливо враховувати не лише факт існування ресурсу, а й його поточний стан. Статистика у предметній області використовується для узагальнення інформації про

ресурси. Вона може показувати загальну кількість об’єктів, кількість доступних ресурсів, ресурси на ремонті, кількість активних заявок або розподіл ресурсів за категоріями. Такі дані допомагають відповідальним особам швидко оцінювати поточний стан обліку та приймати рішення [7].

Отже, предметна область системи ResourceFlow охоплює не тільки зберігання списку ресурсів, а й повний набір пов’язаних понять: категорії, статуси, заявки, ролі, інвентарні коди, місця зберігання, відповідальних осіб, історію змін, доступність ресурсів і статистику. Саме ці поняття формують основу для подальшого проєктування структури даних і функціональності системи.

1.3. Сучасні технології та тенденції, що застосовуються в даній області

У сфері обліку ресурсів усе частіше використовуються веб-орієнтовані інформаційні системи, які дозволяють працювати з даними через браузер без встановлення окремого програмного забезпечення. Такий підхід є зручним для організацій, пунктів прокату, навчальних закладів і складів, оскільки користувач може отримати доступ до системи з комп’ютера, ноутбука або мобільного пристрою. Веб-доступ також спрощує супровід системи, адже оновлення виконуються на рівні проєкту, а не окремо на кожному робочому місці [7].

Однією з важливих тенденцій є централізоване збереження даних. Замість локальних таблиць або паперових журналів сучасні системи використовують бази даних або серверні сховища, у яких зберігається інформація про ресурси, користувачів, заявки, статуси та історію змін. Це дозволяє уникнути дублювання інформації та забезпечити доступ до актуального стану ресурсів. Для системи обліку це особливо важливо, оскільки користувачі повинні бачити однакові й оновлені дані незалежно від пристрою.

Також важливу роль відіграють авторизація та рольовий доступ. Різні користувачі повинні мати різні права: адміністратор керує системою, менеджер

працює із заявками та ресурсами, а звичайний користувач переглядає доступні ресурси або створює заявку [8]. Такий підхід підвищує безпеку системи та зменшує ймовірність випадкових змін або видалення даних.

Ще однією поширеною тенденцією є використання адаптивного дизайну. Оскільки користувачі можуть працювати із системою не лише з комп'ютера, а й зі смартфона, інтерфейс повинен коректно відображатися на різних розмірах екрана. Для цього застосовуються гнучкі сітки, адаптивні таблиці, зручні форми та елементи керування. Це дає змогу використовувати систему безпосередньо під час перевірки, видачі або повернення ресурсу.

На рисунку 1.1 наведено загальну тенденцію розвитку засобів обліку ресурсів від ручних способів до універсальної веб-орієнтованої системи. Такий розвиток пояснюється потребою у швидкому доступі до даних, прозорості процесів і можливості працювати з ресурсами незалежно від конкретного робочого місця.

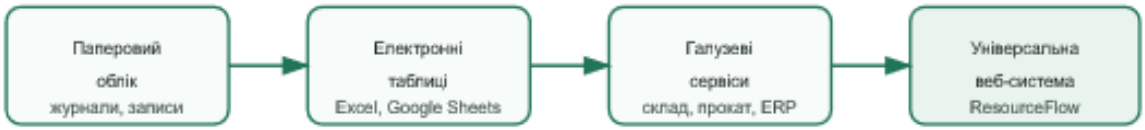


Рисунок 1.1 – Тенденція розвитку засобів обліку ресурсів

У системах обліку ресурсів також активно застосовуються пошук, фільтрація, статуси та аналітичні показники. Користувач повинен мати можливість швидко знайти потрібний ресурс за назвою, категорією або станом. Статистика допомагає відповідальним особам бачити загальну кількість ресурсів, кількість заявок і поточний стан використання майна.

Серед сучасних підходів також можна виділити використання серверних API. Завдяки API клієнтська частина системи не працює з даними напряму, а надсилає запити до серверної частини. Це дозволяє краще розділити відповідальність між інтерфейсом, логікою обробки та збереженням інформації.

Такий підхід є типовим для веб-орієнтованих систем і спрощує подальший розвиток проєкту.

Для невеликих і навчальних систем може застосовуватися файлова модель збереження даних, наприклад JSON-файл. Вона не замінює повноцінну систему керування базами даних, але дозволяє продемонструвати принцип централізованого збереження інформації на сервері. У подальшому таке сховище можна замінити на SQLite, PostgreSQL або іншу СУБД без повної зміни логіки інтерфейсу.

Окреме значення має використання клієнтських технологій HTML, CSS і JavaScript. HTML формує структуру сторінок, CSS відповідає за візуальне оформлення та адаптивність, а JavaScript забезпечує динамічну роботу інтерфейсу. Завдяки цьому користувач може перемикатися між розділами, працювати з формами, фільтрами, таблицями та заявками без складних додаткових інструментів.

Ще однією тенденцією є спрощення інтерфейсів для щоденної роботи. Користувачі таких систем зазвичай не працюють із програмою постійно, тому інтерфейс має бути зрозумілим без складного навчання. Для цього застосовуються зрозумілі таблиці, фільтри, статусні позначки, короткі форми, модальні вікна та логічне групування сторінок.

У сучасних веб-системах важливою є можливість швидкого розгортання. Якщо застосунок можна запустити локально або перенести на VPS-сервер без складної інфраструктури, це підвищує його практичну цінність. Для ResourceFlow це означає, що система може бути доступною через браузер, а дані можуть зберігатися незалежно від конкретного комп’ютера користувача.

Також актуальною є універсальність системи. Замість створення окремого рішення лише для складу, лише для прокату або лише для офісної техніки доцільно передбачити гнучкі категорії та загальні властивості ресурсу. Завдяки цьому одна система може використовуватися в різних сферах без суттєвої зміни структури, основної логіки роботи, принципів обліку та подальшого супроводу [9].

Крім того, сучасні системи обліку повинні підтримувати прозорість дій користувачів. Для цього використовується історія змін, яка фіксує додавання, редагування, зміну статусу або створення заявки. Наявність такого журналу допомагає відповідальним особам краще контролювати роботу з ресурсами та швидше знаходити причини помилок у даних.

Таким чином, сучасні технології й тенденції у сфері обліку ресурсів спрямовані на веб-доступ, централізацію даних, рольовий доступ, адаптивність, просту інтеграцію та зрозумілу взаємодію з користувачем. Усі ці підходи враховано під час формування ідеї та подальшого проєктування системи ResourceFlow.

1.4. Порівняльний аналіз підходів до побудови систем обліку ресурсів

Після розгляду основних понять предметної області та сучасних технологій доцільно порівняти можливі підходи до побудови системи обліку ресурсів. Такий аналіз дозволяє обґрунтувати вибір саме веб-орієнтованої системи, а не паперового журналу, електронної таблиці або складного галузевого програмного продукту.

Найпростішим способом є паперовий облік, коли інформація про ресурси фіксується вручну. Його перевагою є доступність і відсутність потреби у технічній інфраструктурі. Проте такий підхід ускладнює пошук, не підтримує історію змін, не дає змоги швидко отримувати статистику і залежить від уважності відповідальної особи [19].

Електронні таблиці частково вирішують проблему зручності, оскільки дозволяють швидко створити перелік ресурсів, сортувати дані та застосовувати базові фільтри. Однак вони не забезпечують повноцінного розмежування прав доступу, роботи із заявками, контролю статусів і прозорого журналу дій. Через це таблиці зручні лише на початковому етапі, коли кількість ресурсів є невеликою. Коли обсяг даних зростає, підтримувати актуальність такої таблиці стає значно складніше.

Галузеві системи можуть мати розвинений функціонал для складу, прокату або підприємства. Вони часто містять складні довідники, звіти та інтеграції, але для невеликої організації або навчального проєкту можуть бути надмірними. Їх впровадження потребує більше часу, налаштування та підготовки користувачів.

Для ResourceFlow найбільш доцільним є підхід універсальної веб-системи. Він дозволяє працювати через браузер, вести каталог ресурсів, використовувати категорії, статуси, заявки, статистику та історію змін. Така система не прив’язана до однієї галузі, тому може застосовуватися для техніки, інструментів, меблів, транспорту, матеріалів або прокатного обладнання [10].

У таблиці 1.2 наведено порівняння основних підходів до побудови системи обліку ресурсів. Воно показує, що традиційні способи можуть бути простими, але не забезпечують достатнього рівня контролю, тоді як універсальна веб-система краще відповідає задачам ResourceFlow.

Таблиця 1.2

Порівняння підходів до побудови системи обліку ресурсів

Підхід	**Переваги**	**Обмеження**	**Доцільність для ResourceFlow**
Паперовий облік	Простий спосіб фіксації ресурсів без технічної інфраструктури.	Складний пошук, відсутність статистики, історії змін і одночасної роботи кількох користувачів.	Не відповідає вимогам сучасної веб-системи.
Електронні таблиці	Дають змогу швидко створити перелік ресурсів і виконувати базову фільтрацію.	Не забезпечують повноцінних ролей доступу, заявок і контрольованої історії дій.	Підходять лише для початкового або дуже простого обліку.
Галузеві системи	Можуть мати розвинені модулі складу, оренди або управління майном.	Часто є складними, дорогими або прив’язаними до конкретної сфери діяльності.	Можуть бути надлишковими для невеликої організації.
Універсальна веб-система	Підтримує браузерний доступ, категорії, статуси, заявки, статистику і збереження даних.	Потребує проєктування інтерфейсу, серверної частини та структури даних.	Найкраще відповідає меті системи ResourceFlow.

Як видно з таблиці 1.2, паперовий облік і електронні таблиці можуть використовуватися лише для нескладних задач. Вони не забезпечують достатньої автоматизації, прозорості та контролю. Галузеві системи, навпаки, можуть бути занадто складними для поставленої задачі. Тому універсальна веб-система є збалансованим рішенням, яке поєднує простоту використання з необхідною функціональністю.

Окремо варто врахувати зручність щоденного використання. Якщо система є занадто складною, користувачі можуть уникати її застосування або продовжувати вести облік у таблицях. Тому для ResourceFlow важливо, щоб основні дії були очевидними: додати ресурс, змінити статус, знайти потрібний об’єкт, створити заявку або переглянути статистику [11].

Перевагою веб-орієнтованого підходу є також можливість доступу з різних пристроїв. Користувачу не потрібно встановлювати окрему програму, оскільки робота виконується через браузер. Це зручно для ситуацій, коли ресурс потрібно перевірити безпосередньо під час видачі, повернення або інвентаризації.

Для системи обліку ресурсів важливим є не лише збереження списку об’єктів, а й підтримка логіки їх використання. Саме тому ResourceFlow поєднує каталог ресурсів, категорії, статуси, заявки та історію змін. Такий підхід дозволяє розглядати ресурс не як окремий рядок у таблиці, а як об’єкт, який має стан, відповідальну особу та зв’язок із діями користувачів.

Також важливо, щоб обрана система не обмежувала користувача однією сферою застосування. ResourceFlow може використовуватися для обліку офісної техніки, інструментів, матеріалів, меблів або прокатного обладнання. Це досягається завдяки загальній структурі ресурсу та можливості створювати власні категорії. У результаті система залишається гнучкою навіть при зміні потреб організації.

Важливим критерієм вибору є також можливість подальшого розвитку. ResourceFlow може поступово доповнюватися новими функціями, наприклад календарем бронювання, розширеними звітами, автоматичними повідомленнями

або переходом на повноцінну базу даних. Це робить обраний підхід гнучким і придатним для вдосконалення [19].

Таким чином, порівняльний аналіз підтверджує доцільність створення саме веб-орієнтованої інформаційної системи обліку ресурсів. Такий підхід відповідає меті роботи, оскільки дозволяє створити універсальний, зрозумілий і достатньо функціональний інструмент для контролю ресурсів у різних умовах використання.

1.5. Висновки по розділу

У першому розділі було розглянуто сучасний стан проблематики обліку ресурсів і визначено, що традиційні способи ведення обліку не завжди забезпечують зручність, швидкість пошуку, контроль статусів та історію змін. Було встановлено, що для ефективної роботи потрібна система, яка дозволяє централізовано зберігати інформацію про ресурси, користувачів, заявки та відповідальних осіб. Також було визначено основні поняття предметної області: ресурс, категорія, статус, заявка, користувач і роль доступу. Ці поняття формують основу подальшого проєктування системи ResourceFlow, оскільки саме навколо них будується логіка обліку, видачі, пошуку та аналізу ресурсів. Окрему увагу приділено універсальності поняття ресурсу, оскільки система має підтримувати різні типи об’єктів без зміни основної структури.

Крім того, було розглянуто сучасні технології та тенденції, що застосовуються у сфері обліку ресурсів, а також виконано порівняльний аналіз підходів до побудови таких систем. До важливих напрямів належать веб-орієнтований доступ, централізоване збереження даних, рольовий доступ, адаптивний дизайн, пошук, фільтрація та статистика. Отже, результати першого розділу підтверджують доцільність створення універсальної веб-системи ResourceFlow для обліку ресурсів у різних сферах діяльності. Отримані результати стали теоретичною основою для формування вимог і постановки задачі проєктування системи у наступному розділі.

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ

2.1. Збір та аналіз вимог користувачів

Під час визначення вимог було виділено три основні ролі користувачів: адміністратор, працівник або менеджер і звичайний користувач. Адміністратор відповідає за повне керування системою: додавання, редагування та видалення ресурсів, категорій і користувачів. Працівник або менеджер працює з ресурсами й заявками, контролює статуси та переглядає звіти. Звичайний користувач має обмежені можливості: перегляд доступних ресурсів і створення заявки.

Такий розподіл ролей відповідає реальній логіці організації процесу. У пункті прокату власник або старший адміністратор не повинен виконувати всі дрібні операції самостійно, але має контролювати довідники та права доступу. Менеджер може приймати заявки й оновлювати статуси, але не обов’язково повинен видаляти категорії або змінювати користувачів. Користувачеві достатньо бачити каталог і ініціювати запит на ресурс [10].

Основними сценаріями використання є вхід або реєстрація, перегляд головної панелі, пошук ресурсу, фільтрація за категорією або статусом, додавання нового ресурсу, редагування картки ресурсу, створення заявки, погодження або відхилення заявки, перегляд статистики, експорт звіту та редагування профілю. Кожен сценарій повинен мати зрозумілий шлях у навігації, оскільки система орієнтована на користувачів без спеціальної технічної підготовки.

Сценарій додавання ресурсу є одним із центральних. Користувач із правами адміністратора відкриває форму, заповнює основні дані, обирає категорію або створює нову, визначає статус і за потреби додає фото. Фото не є обов’язковим, тому відсутність зображення не блокує збереження. Це важливо, бо під час інвентаризації не завжди є можливість одразу зробити фотографію кожного ресурсу. Сценарій створення заявки пов’язує ресурс із конкретною потребою. Користувач обирає ресурс, вказує призначення та створює запис. Далі

менеджер або адміністратор змінює статус заявки. Такий процес робить використання ресурсів контрольованим і дозволяє аналізувати не тільки самі ресурси, а й попит на них.

У результаті аналізу сценаріїв було сформовано основу для інтерфейсу: ліве меню з основними розділами, головна панель зі статистикою, каталог із фільтрами, окремі сторінки для форм, категорій, заявок, звітів, користувачів і профілю. Це забезпечує логічну структуру, яка відповідає очікуванням користувача.

Додатково у межах цього підрозділу важливо врахувати відокремлення прав користувачів від їх щоденних сценаріїв. Один і той самий ресурс може цікавити різних учасників: користувач хоче подати заявку, менеджер — погодити її, а адміністратор — змінити дані або категорію. Через це під час аналізу було зроблено акцент не лише на формальному переліку функцій, а й на тому, як ці функції підтримують реальний робочий процес користувача. Такий підхід дозволяє уникнути перевантаження інтерфейсу зайвими діями для окремих ролей.

Практичне значення такого підходу полягає в тому, що система не нав’язує користувачеві жорстку галузеву модель. Вона залишає можливість описувати різні категорії, змінювати склад ресурсів, працювати з різними відповідальними особами та поступово нарощувати обсяг даних без зміни основної логіки застосунку.

Саме тому сценарії використання були описані окремо для кожної ролі, а інтерфейс показує тільки ті дії, які мають практичний сенс для поточного користувача [11]. Це рішення узгоджується з метою бакалаврської роботи, оскільки демонструє не ізольований екран або окрему форму, а послідовно спроєктовану інформаційну систему з визначеною структурою даних, ролями доступу, інтерфейсом і механізмами збереження інформації.

На рисунку 2.1 наведено діаграму варіантів використання системи ResourceFlow, яка відображає основні дії адміністратора, менеджера та користувача.

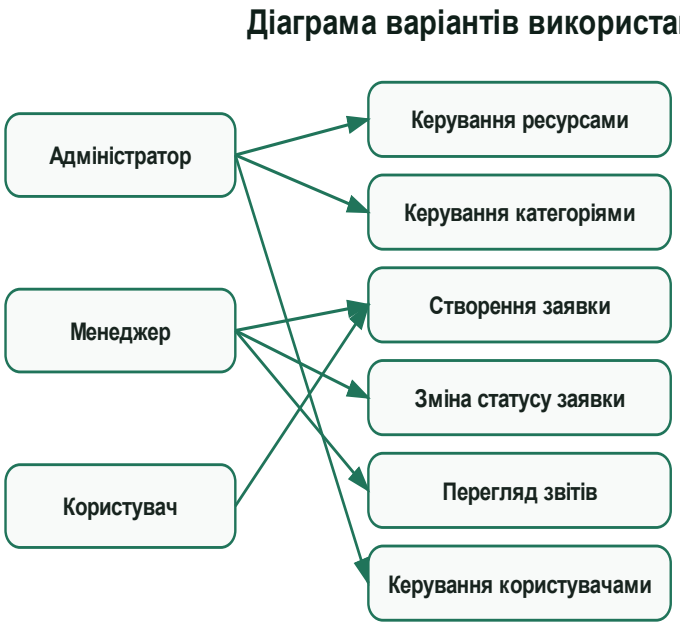


Рисунок 2.1 – Діаграма варіантів використання системи

2.2. Функціональні та нефункціональні вимоги

Після визначення основних користувачів системи було сформовано функціональні та нефункціональні вимоги до веб-орієнтованої інформаційної системи ResourceFlow. Функціональні вимоги описують дії, які система повинна виконувати, а нефункціональні визначають якість її роботи, зручність, адаптивність і надійність. Поєднання цих двох груп вимог дозволяє розглядати систему не лише як набір окремих функцій, а як повноцінний інструмент для щоденного обліку ресурсів. Саме вимоги визначають, які можливості мають бути реалізовані в першу чергу.

До основних функціональних вимог належить авторизація та реєстрація користувачів. Система повинна дозволяти користувачу створити обліковий запис і виконати вхід за допомогою email та пароля. Також необхідно передбачити розмежування прав доступу залежно від ролі користувача: адміністратора, менеджера або звичайного користувача. Це потрібно для того, щоб кожен користувач працював тільки з тими функціями, які відповідають його повноваженням [12].

Важливою функціональною вимогою є облік ресурсів. Система повинна забезпечувати додавання, редагування, видалення та перегляд ресурсів. Для кожного ресурсу необхідно зберігати назву, категорію, інвентарний код, місце зберігання, відповідальну особу, статус, вартість і за потреби фото. Крім того, користувач повинен мати можливість виконувати пошук ресурсів, фільтрувати їх за категорією або статусом і отримувати потрібну інформацію з каталогу. Це дозволяє швидко працювати навіть із великою кількістю записів.

Окрему групу функцій становить робота із заявками. Користувач повинен мати можливість створити заявку на використання ресурсу, а менеджер або адміністратор – змінити її статус. Також система повинна зберігати історію основних дій, пов’язаних із ресурсами та заявками. Це дозволяє відстежувати зміни, контролювати процес використання ресурсів і зменшувати ризик втрати важливої інформації.

До функціональних вимог також належать керування категоріями, перегляд статистики та формування звітної інформації. Категорії дозволяють групувати ресурси за типом, наприклад техніка, транспорт, інструменти, матеріали або меблі. Статистика повинна показувати загальну кількість ресурсів, доступні ресурси, заявки та ресурси, що перебувають на ремонті. Такі показники допомагають швидко оцінити стан системи та приймати управлінські рішення.

Нефункціональні вимоги визначають якість роботи системи. Насамперед система повинна мати зручний і зрозумілий інтерфейс, щоб користувач міг виконувати основні дії без спеціальної підготовки. Сторінки мають бути логічно організовані, а форми – простими для заповнення.

Ще однією нефункціональною вимогою є адаптивність. Система повинна коректно відображатися на різних пристроях, зокрема на комп’ютері, ноутбуці, планшеті та телефоні. Для цього інтерфейс має змінювати розміщення блоків залежно від ширини екрана, а таблиці та форми повинні залишатися зручними для перегляду й заповнення. Це важливо, оскільки облік ресурсів може виконуватися не лише з робочого місця, а й безпосередньо під час перевірки або видачі майна.

Також до нефункціональних вимог належать збереження даних, стабільність роботи та простота розгортання. Інформація про ресурси, користувачів, заявки й категорії повинна зберігатися після виходу із системи. Серверна частина має забезпечувати читання та запис даних без втрати інформації. Крім того, структура проєкту повинна залишатися достатньо простою, щоб систему можна було запустити локально або перенести на сервер без складного налаштування.

Отже, сформовані функціональні та нефункціональні вимоги визначають основні можливості й якісні характеристики системи ResourceFlow. Вони охоплюють авторизацію, рольовий доступ, CRUD-операції з ресурсами, категорії, заявки, історію змін, статистику, адаптивний інтерфейс і збереження даних. Саме ці вимоги стали основою для подальшого проєктування архітектури, моделі даних і програмної реалізації системи.

У таблиці 2.1 наведено основні функціональні вимоги до системи ResourceFlow, сформовані на основі аналізу ролей користувачів і типових сценаріїв використання. Вона демонструє, які дії повинна забезпечувати система для підтримки процесу обліку ресурсів, контролю їхнього стану та організації заявок на використання.

Таблиця 2.1

Основні функціональні вимоги до системи

№	Функціональна вимога	Опис
1	Авторизація та реєстрація	Користувач повинен мати можливість увійти в систему або створити новий обліковий запис.
2	Розмежування прав доступу	Система повинна враховувати ролі адміністратора, менеджера та користувача.
3	Облік ресурсів	Користувачі з відповідними правами повинні мати можливість додавати, редагувати, видаляти та переглядати ресурси.
4	Керування категоріями	Система повинна дозволяти створювати, змінювати та видаляти категорії ресурсів.
5	Пошук і фільтрація	Користувач повинен мати можливість швидко знаходити ресурси за назвою, категорією або статусом.
6	Робота із заявками	Система повинна дозволяти створювати заявки на використання ресурсів та змінювати їхній статус.
7	Історія змін	Система повинна зберігати інформацію про основні дії користувачів із ресурсами та заявками.

2.3. Постановка задачі проєктування системи

Після аналізу предметної області та визначення основних вимог було сформовано задачу проєктування веб-орієнтованої інформаційної системи обліку ресурсів ResourceFlow. Основна мета проєктування полягає у створенні системи, яка дозволяє вести облік різних типів ресурсів, контролювати їхній стан, відповідальних осіб, заявки на використання та історію змін. Система повинна бути універсальною, тобто придатною не лише для одного виду майна, а для техніки, обладнання, меблів, інструментів, транспорту, матеріалів або програмних ліцензій.

У межах проєктування необхідно передбачити структуру системи, яка включає клієнтську частину, серверну частину та окреме сховище даних. Клієнтська частина повинна забезпечувати зручну взаємодію користувача із системою через браузер [13]. Серверна частина має відповідати за обробку запитів, читання та запис даних. Сховище даних повинно містити інформацію про ресурси, категорії, користувачів, заявки та історію змін.

На рисунку 2.2 наведено основні складові задачі проєктування системи ResourceFlow. Схема показує, що під час проєктування необхідно врахувати інтерфейс користувача, серверну частину, сховище даних, рольовий доступ, заявки, статистику та можливість подальшого розвитку системи.

Рисунок 2.2 – Основні складові задачі проєктування системи ResourceFlow

Одним із важливих завдань є реалізація авторизації та розмежування доступу. У системі повинні бути передбачені різні ролі користувачів: адміністратор, менеджер і звичайний користувач. Адміністратор повинен мати можливість керувати ресурсами, категоріями та користувачами. Менеджер має працювати з ресурсами й заявками. Користувач повинен мати доступ до перегляду ресурсів і створення заявки.

Також необхідно передбачити реалізацію основних функцій системи: додавання, редагування та видалення ресурсів, керування категоріями, пошук і фільтрацію, створення заявок, зміну статусів, перегляд статистики та історії змін. Для кожного ресурсу потрібно зберігати назву, категорію, інвентарний код, місце зберігання, відповідальну особу, статус, вартість і за потреби фото.

Окремою задачею є забезпечення зручності інтерфейсу та адаптивності. Система повинна коректно відображатися як на екрані комп'ютера, так і на мобільному пристрої. Основні сторінки мають бути логічно згруповані: авторизація, головна панель, каталог ресурсів, додавання та редагування ресурсу, категорії, заявки, звіти, користувачі та профіль [14]. Такий поділ сторінок робить навігацію зрозумілою та послідовною.

Важливою частиною задачі є забезпечення цілісності даних. Ресурси, категорії, заявки, користувачі та історія змін повинні бути пов'язані між собою логічно. Наприклад, ресурс має належати до певної категорії, заявка повинна стосуватися конкретного ресурсу, а історія змін має відображати важливі дії користувачів.

Під час постановки задачі також потрібно врахувати можливість подальшого розвитку системи. На початковому етапі достатньо реалізувати базові функції обліку, але структура проєкту повинна дозволяти додати календар бронювання, експорт звітів, повідомлення користувачів або перехід на повноцінну базу даних без повного переписування системи.

У межах системи необхідно забезпечити базову аналітику. Статистика повинна показувати загальну кількість ресурсів, кількість доступних ресурсів, заявки, що очікують розгляду, та ресурси, які перебувають на ремонті.Такі

показники дозволяють відповідальній особі швидко оцінювати стан системи без ручного підрахунку даних у каталозі.

Отже, задача проєктування полягає у створенні універсальної веб-системи, яка поєднує облік ресурсів, рольовий доступ, роботу із заявками, статистику та збереження даних. Реалізація такої системи дозволить упорядкувати процес контролю ресурсів, зменшити кількість ручної роботи та забезпечити зручний доступ до актуальної інформації через браузер.

2.4. Висновки по розділу

У другому розділі було визначено основних користувачів системи ResourceFlow та описано їхні сценарії роботи. Розглянуто ролі адміністратора, менеджера і користувача, а також їхню взаємодію з ресурсами, категоріями, заявками та звітами. Такий розподіл дозволив зрозуміти, які функції повинні бути доступні кожній ролі та як система має підтримувати щоденну роботу з ресурсами.

Також було сформовано функціональні та нефункціональні вимоги до системи. Вони охоплюють авторизацію, рольовий доступ, облік ресурсів, керування категоріями, роботу із заявками, пошук, фільтрацію, статистику, адаптивність інтерфейсу та збереження даних. Окрему увагу приділено зручності використання системи, оскільки ResourceFlow має бути зрозумілою для користувачів без спеціальної технічної підготовки.

У результаті сформовано постановку задачі проєктування системи, яка полягає у створенні універсального веб-застосунку для обліку ресурсів із рольовим доступом, зручною структурою, серверною частиною та можливістю збереження інформації. Визначено, що система повинна підтримувати роботу з різними типами ресурсів, заявками, статусами, історією змін і базовою статистикою. Сформовані вимоги стали основою для подальшого вибору архітектури, моделі даних, загальної структури застосунку та технологій реалізації системи ResourceFlow.

РОЗДІЛ 3. ПРОЄКТУВАННЯ СИСТЕМИ ОБЛІКУ РЕСУРСІВ

3.1. Архітектурна модель системи обліку ресурсів

Архітектура ResourceFlow побудована як легкий веб-застосунок, що складається з клієнтської частини, серверного API та файлової бази даних. Клієнтська частина розміщена у папці public і містить HTML-сторінку, таблицю стилів та JavaScript-файл із логікою інтерфейсу. Серверна частина реалізована на Node.js і відповідає за видачу статичних файлів та роботу з маршрутом /api/db.

Для кращого розуміння взаємодії між основними частинами системи було побудовано схему архітектури ResourceFlow. Вона відображає зв'язок між браузером користувача, клієнтською частиною, серверним API та файловою базою даних. На рисунку 3.1 наведено архітектуру веб-орієнтованої системи, яка демонструє проходження даних від інтерфейсу користувача до серверної частини та подальшого збереження у файлі бази даних [15].

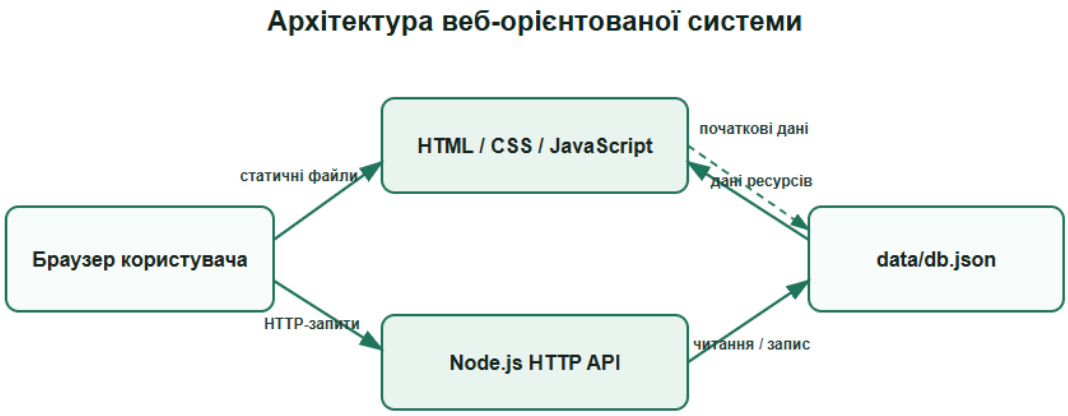


Рисунок 3.1 – Архітектура веб-орієнтованої системи

Вибір такої архітектури пояснюється навчальною метою і практичністю. Важливо показати повний цикл: від взаємодії користувача з браузером до збереження даних на сервері. При цьому використання складного фреймворку не є обов'язковим, оскільки основна логіка може бути реалізована чистими засобами HTML, CSS, JavaScript і стандартних модулів Node.js.

Клієнтська частина працює як односторінковий застосунок. Стан системи зберігається в об’єкті state, а зміна сторінки або даних призводить до повторного рендерингу відповідного шаблону. Такий підхід дозволяє швидко перемикатися між розділами без перезавантаження HTML-документа. Для користувача це виглядає як повноцінний веб-інтерфейс із меню, формами, таблицями і модальними вікнами [16].

Серверна частина має дві відповідальності. По-перше, вона віддає файли інтерфейсу: index.html, styles.css, app.js та інші статичні ресурси. По-друге, вона обробляє GET і PUT запити до /api/db. GET повертає поточний JSON із базою даних, а PUT приймає оновлений JSON і записує його у файл. Це проста, але зрозуміла модель персистентності.

Окреме значення має структура проєкту. Вона поділяє публічні файли, сервер, базу даних і документацію. Такий поділ робить проєкт більш професійним і зрозумілим для подальшого супроводу. У майбутньому цю архітектуру можна розширити: замінити JSON-файл на PostgreSQL або SQLite, додати окремі REST-маршрути, middleware для авторизації та резервне копіювання [17].

Архітектурна модель відповідає принципу достатності, вона не перевантажує навчальний прототип, але демонструє ключові компоненти реальної веб-системи. Саме цей баланс дозволив реалізувати потрібні функції у компактному проєкті.

Додатково у межах цього підрозділу важливо врахувати простота архітектури як перевага навчального і демонстраційного проєкту. У системі немає зайвого проміжного шару або складного набору залежностей, тому шлях даних легко простежити: браузер формує дію, JavaScript оновлює стан, сервер приймає JSON і записує його у файл [18]. Через це під час аналізу було зроблено акцент не лише на формальному переліку функцій, а й на тому, як ці функції підтримують реальний робочий процес користувача.

Практичне значення такого підходу полягає в тому, що система не нав’язує користувачеві жорстку галузеву модель. Вона залишає можливість

описувати різні категорії, змінювати склад ресурсів, працювати з різними відповідальними особами та поступово нарощувати обсяг даних без зміни основної логіки застосунку.

Це робить архітектуру прозорою для аналізу, а також дає змогу надалі замінити окремі частини без повного переписування системи. Це рішення узгоджується з метою бакалаврської роботи, оскільки демонструє не ізольований екран або окрему форму, а послідовно спроєктовану інформаційну систему з визначеною структурою даних, ролями доступу, інтерфейсом і механізмами збереження інформації.

3.2. Моделювання бізнес-процесів та системних компонентів

Моделювання бізнес-процесів є важливим етапом проєктування інформаційної системи, оскільки воно дозволяє визначити, які дії виконують користувачі, які дані обробляються системою та які результати мають бути отримані після виконання кожного сценарію. Для системи ResourceFlow основним бізнес-процесом є облік ресурсу від моменту його додавання до каталогу до подальшого використання, зміни статусу, формування заявки або перегляду статистики.

У межах системи можна виділити кілька основних процесів: реєстрація та авторизація користувача, додавання ресурсу, редагування даних ресурсу, керування категоріями, створення заявки, зміна статусу заявки, перегляд історії змін і формування звітів. Кожен із цих процесів пов’язаний з окремими компонентами системи, але всі вони працюють з єдиним набором даних. Це забезпечує цілісність роботи системи та дозволяє уникнути дублювання інформації [11].

Процес додавання ресурсу починається з того, що адміністратор або менеджер відкриває форму створення нового ресурсу. Після цього користувач заповнює основні поля: назву, категорію, інвентарний код, місце зберігання, відповідальну особу, статус, вартість і за потреби додає фото. Після збереження

дані передаються через клієнтську частину до серверного API, після чого записуються у файл бази даних data/db.json. У результаті новий ресурс з'являється в каталозі та може бути використаний у заявках або статистиці.

Процес створення заявки пов'язує користувача з конкретним ресурсом. Користувач обирає потрібний ресурс, вказує мету використання та створює заявку. Система зберігає заявку зі статусом очікування, після чого менеджер або адміністратор може її погодити чи відхилити. У разі зміни статусу заявки відповідна подія додається до історії змін. Така модель дозволяє контролювати не тільки наявність ресурсу, а й процес його видачі або використання.

Системні компоненти ResourceFlow можна поділити на клієнтські, серверні та компоненти збереження даних. До клієнтських компонентів належать сторінки інтерфейсу, форми, таблиці, фільтри, меню та модальні вікна. Вони відповідають за взаємодію користувача із системою. Серверний компонент представлений Node.js HTTP API, який приймає запити від клієнта, читає дані з бази та записує оновлену інформацію. Компонент збереження даних представлений файлом data/db.json, у якому містяться ресурси, категорії, користувачі, заявки та історія. Завдяки цьому дані залишаються доступними після завершення сеансу роботи.

Взаємодія компонентів відбувається за простою схемою. Користувач виконує дію в браузері, клієнтська частина формує запит до сервера, сервер обробляє цей запит і звертається до файлу бази даних. Після цього оновлені дані повертаються до клієнтської частини та відображаються в інтерфейсі. Такий підхід дозволяє розділити відповідальність між частинами системи: інтерфейс відповідає за зручність роботи, сервер – за обробку запитів, а база даних – за збереження інформації [22].

Таким чином, моделювання бізнес-процесів і системних компонентів дозволило визначити логіку роботи ResourceFlow та зв'язки між основними частинами системи. Запропонована модель є достатньо простою для реалізації, але водночас охоплює основні сценарії використання системи: облік ресурсів, роботу із заявками, зміну статусів, історію змін і формування звітної інформації.

3.3. Проєктування бази даних та обґрунтування технологій

Для реалізації веб-орієнтованої інформаційної системи ResourceFlow було обрано набір технологій, який дозволяє створити простий, зрозумілий і достатньо функціональний веб-застосунок. Основною вимогою до технологічного стеку була можливість швидкої розробки інтерфейсу, обробки даних на сервері та збереження інформації між сеансами роботи користувача. Оскільки система призначена для демонстрації обліку ресурсів, важливо було використати такі інструменти, які не ускладнюють архітектуру, але дають змогу реалізувати авторизацію, роботу з ресурсами, категоріями, заявками, історією змін і статистикою.

Клієнтська частина системи реалізована за допомогою HTML, CSS і JavaScript. HTML використовується для формування структури сторінок, CSS відповідає за зовнішній вигляд, адаптивність і розміщення елементів інтерфейсу, а JavaScript забезпечує динамічну роботу сторінок. За допомогою JavaScript реалізовано перемикання між розділами системи, обробку форм, пошук, фільтрацію, зміну статусів, відображення таблиць, роботу з фото ресурсу та взаємодію із сервером. Такий підхід є доцільним для бакалаврського проєкту, оскільки дозволяє показати логіку роботи веб-застосунку без використання надмірно складних фреймворків [4].

Для узагальнення обраного технологічного стеку було сформовано таблицю з основними інструментами, їх призначенням у проєкті та причинами вибору. Такий формат дозволяє коротко показати, яку роль виконує кожна технологія в системі ResourceFlow і чому саме вона є доцільною для реалізації навчального веб-застосунку. Крім того, обрані технології є доступними для локального запуску та подальшого розгортання системи на сервері без складної інфраструктури.

У таблиці 3.1 наведено обґрунтування вибору технологій розробки, що використовуються для створення клієнтської частини, серверної частини та збереження даних.

Таблиця 3.1

Обґрунтування вибору технологій розробки

Технологія або інструмент	**Призначення у проєкті**	**Обґрунтування вибору**
HTML	Структура веб-сторінок	Дозволяє створити основу інтерфейсу системи та розмістити основні елементи сторінок.
CSS	Оформлення та адаптивність	Забезпечує зовнішній вигляд системи, розміщення блоків і коректне відображення на різних екранах.
JavaScript	Логіка клієнтської частини	Використовується для роботи форм, таблиць, фільтрів, пошуку та взаємодії з сервером.
Node.js	Серверна частина системи	Дозволяє обробляти HTTP-запити та організувати роботу з даними системи.
JSON-файл	Збереження даних	Використовується як проста файлова база даних для ресурсів, користувачів, заявок і категорій.
Браузер	Середовище роботи користувача	Дає змогу користуватися системою без встановлення додаткового програмного забезпечення.

Для серверної частини було обрано Node.js. Ця технологія дає змогу створити легкий HTTP-сервер, який обробляє запити від клієнтської частини та працює з даними системи. Сервер відповідає за отримання, оновлення та збереження інформації про ресурси, користувачів, категорії, заявки й історію змін. Використання Node.js є зручним, оскільки і клієнтська, і серверна логіка пишуться мовою JavaScript. Це спрощує розробку, зменшує кількість різних технологій у проєкті і робить код більш зрозумілим для підтримки.

Для збереження даних у системі використовується окремий файл data/db.json. У ньому зберігаються основні сутності системи: користувачі, ресурси, категорії, заявки та історія змін [25]. На відміну від збереження інформації тільки в браузері, файлова база даних не зникає після виходу користувача із сайту і може використовуватися серверною частиною для подальшого читання та запису.

Окрему увагу було приділено адаптивності інтерфейсу. Система повинна бути зручною не лише на комп’ютері, а й на пристроях із меншою шириною екрана. Для цього в CSS використано гнучкі сітки, адаптивні блоки, зміну

кількості колонок і коректне відображення таблиць. Це дозволяє користувачу працювати з ресурсами, заявками та формами навіть на мобільному пристрої. Такий підхід підвищує практичну цінність системи, оскільки облік ресурсів може виконуватися не тільки з робочого місця, а й безпосередньо під час видачі або перевірки майна [28].

Вибір зазначених технологій є обґрунтованим, оскільки вони забезпечують достатню функціональність для реалізації поставлених завдань і водночас не перевантажують проєкт зайвою складністю. HTML, CSS і JavaScript дозволяють створити зручний інтерфейс, Node.js забезпечує серверну обробку запитів, а data/db.json виконує роль простої бази даних для збереження інформації [7].

3.4. Висновки по розділу

У третьому розділі було виконано проєктування інформаційної системи обліку ресурсів ResourceFlow: визначено архітектурну модель системи, змодельовано основні бізнес-процеси та системні компоненти, а також обґрунтовано вибір технологій розробки. Архітектура системи включає браузер користувача, клієнтську частину, серверний HTTP API та файл бази даних data/db.json, що дозволяє розділити інтерфейс, обробку запитів і збереження даних. До основних процесів віднесено додавання й редагування ресурсів, керування категоріями, створення заявок, зміну статусів, перегляд історії та формування звітів.

Також у розділі було визначено, що для реалізації системи доцільно використати HTML, CSS, JavaScript, Node.js і JSON-файл, оскільки цей набір технологій є достатнім для створення простої, адаптивної та функціональної системи. Обрана структура дає змогу реалізувати основні вимоги до ResourceFlow без надмірного ускладнення архітектури та залишає можливість подальшого розширення проєкту. Отже, у третьому розділі сформовано проєктну основу для подальшої програмної реалізації системи ResourceFlow.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЄКТУ ОБЛІКУ РЕСУРСІВ

4.1. Реалізація структури проєкту

Практична реалізація веб-орієнтованої інформаційної системи ResourceFlow виконувалася з урахуванням того, що проєкт має бути зрозумілим для запуску, перевірки та подальшого доопрацювання. Тому структура застосунку була організована так, щоб інтерфейс, серверна логіка, файл даних і допоміжні матеріали не змішувалися між собою. Такий підхід спрощує супровід системи, оскільки розробник або адміністратор може швидко визначити, де знаходиться потрібна частина проєкту.

Основу клієнтської частини становлять файли, які відкриваються у браузері користувача. HTML-документ формує базову структуру сторінки, таблиця стилів відповідає за зовнішній вигляд, а JavaScript-файл забезпечує динамічну поведінку системи. Саме у клієнтській логіці реалізовано перемикання між розділами, обробку форм, побудову таблиць, відображення модальних вікон, роботу з пошуком, фільтрами, заявками, категоріями та ресурсами [13]. Завдяки цьому більшість дій виконується без перезавантаження сторінки.

Серверна частина реалізована як легкий HTTP-сервер на Node.js. Вона виконує дві основні задачі: віддає статичні файли інтерфейсу та забезпечує доступ до даних через API. Такий варіант є достатнім для бакалаврського проєкту, оскільки дозволяє показати повний цикл роботи веб-застосунку без використання надмірно складної серверної архітектури [29].

Для збереження інформації використовується окремий файл data/db.json. У ньому розміщуються основні сутності системи: користувачі, ресурси, категорії, заявки та історія змін. На відміну від збереження даних тільки у браузері, файлова база дозволяє не втрачати інформацію після виходу із системи або перезапуску сторінки. Це важливо для системи обліку, оскільки її головна задача полягає саме у накопиченні та збереженні актуальних записів.

Реалізація API побудована так, щоб клієнтська частина могла отримувати поточний стан бази та передавати оновлені дані після виконання користувацьких дій. Наприклад, після додавання ресурсу JavaScript формує оновлений набір даних і надсилає його на сервер, а сервер записує зміни у файл. Така схема є простою для аналізу і наочно демонструє зв’язок між інтерфейсом та збереженням інформації.

Під час роботи із серверною частиною важливо було забезпечити зрозумілий обмін даними між браузером і файловим сховищем. Користувач не взаємодіє з файлом data/db.json напряму, тому всі зміни проходять через програмну логіку застосунку [31]. Це дозволяє уникнути ситуації, коли користувач випадково змінює структуру файлу або видаляє важливі записи поза інтерфейсом системи. Також це робить процес оновлення даних більш контрольованим.

У межах реалізації було використано підхід, за якого поточний стан системи зберігається у структурованому вигляді. Ресурси мають власні поля, категорії виступають довідником, заявки пов’язують користувача з конкретним ресурсом, а історія змін фіксує важливі події. Така організація даних допомагає уникнути хаотичного зберігання інформації і робить подальшу обробку записів більш передбачуваною.

Суттєвою перевагою такої реалізації є простота перевірки. Файл бази даних можна відкрити і побачити, які саме записи створює система після дій користувача. Для бакалаврської роботи це корисно, оскільки дозволяє продемонструвати не лише зовнішній вигляд сторінок, а й внутрішній результат роботи програмного забезпечення.

Важливою особливістю реалізації є те, що файл бази даних відокремлений від візуального інтерфейсу. Це дозволяє змінювати зовнішній вигляд сторінок без втрати даних, а також спрощує резервне копіювання. Якщо необхідно перенести систему на інший комп’ютер або сервер, достатньо зберегти папку проєкту разом із файлом data/db.json, у якому міститься фактичний стан системи.

Крім того, відокремлення даних від інтерфейсу створює основу для майбутнього розвитку. Якщо в подальшому виникне потреба перейти на SQLite або PostgreSQL, логіку відображення сторінок не потрібно буде повністю переписувати. Достатньо змінити серверний рівень роботи з даними, зберігши загальну поведінку системи для користувача.

Під час створення структури проєкту також враховувалася можливість розгортання на сервері. Для цього важливо, щоб застосунок мав зрозумілу точку запуску, окремі публічні файли та передбачуване місце збереження бази даних. Така організація допомагає уникнути конфліктів з іншими проєктами на сервері та спрощує налаштування окремого порту або служби запуску [33].

На рисунку 4.1 наведено структуру проєкту ResourceFlow у середовищі розробки. Зображення демонструє практичну організацію файлів і підтверджує, що система має окремі частини для інтерфейсу, серверної логіки та збереження даних.

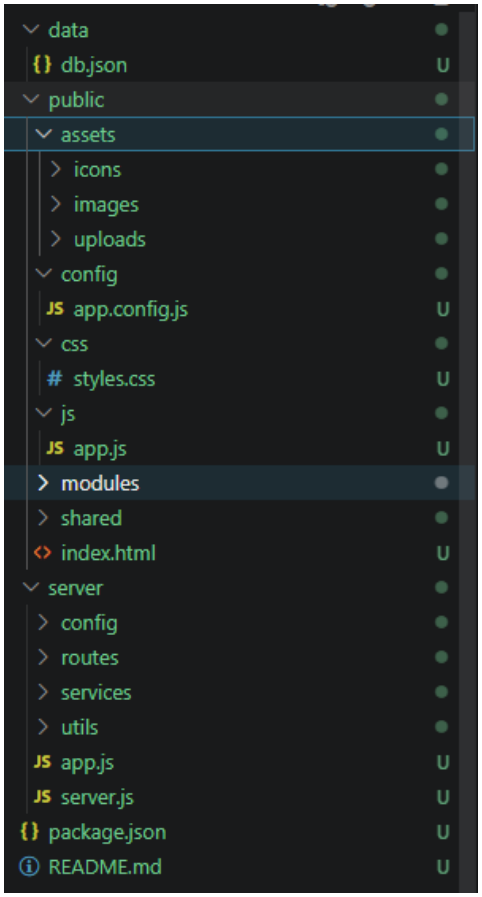


Рисунок 4.1 – Структура реалізованого проєкту ResourceFlow

Під час реалізації було важливо не перевантажувати проєкт зайвими залежностями. Для навчальної роботи доцільно показати, як основні можливості веб-системи можуть бути створені з використанням базових технологій. Тому клієнтська частина не використовує складних фреймворків, а серверна логіка залишається достатньо компактною для аналізу.

Компактність реалізації також полегшує пояснення роботи системи у тексті бакалаврської роботи. Кожен модуль має очевидне призначення, а шлях даних можна простежити від натискання кнопки в інтерфейсі до оновлення запису у файлі бази даних. Це робить проєкт зручним для демонстрації та захисту, оскільки основні рішення не приховані всередині великого фреймворку. Крім того, така структура спрощує подальше внесення змін, оскільки окремі частини системи можна доопрацьовувати без повного переписування всього проєкту.

Для демонстрації складу реалізованих модулів у таблиці 4.1 наведено основні частини системи та їх призначення. Ця таблиця узагальнює програмну реалізацію і показує, які функціональні блоки були створені в межах роботи.

Таблиця 4.1

Основні модулі реалізованої системи ResourceFlow

№	Модуль системи	Реалізоване призначення
1	Авторизація	Вхід, реєстрація та визначення поточної ролі користувача.
2	Головна панель	Відображення загальної статистики, останніх ресурсів і журналу змін.
3	Каталог ресурсів	Перегляд, пошук, фільтрація, редагування та видалення записів.
4	Категорії	Створення, редагування і видалення груп ресурсів.
5	Заявки	Створення запитів на використання ресурсу та зміна їх статусів.
6	Звіти	Узагальнення показників системи та підготовка даних для аналізу.

Як видно з таблиці 4.1, реалізація охоплює не лише каталог ресурсів, а й пов’язані з ним допоміжні частини: авторизацію, категорії, заявки, статистику та історію змін. Це дозволяє користувачу працювати з ресурсами в одному середовищі, не використовуючи окремі таблиці або документи для кожної операції.

Водночас така простота не означає відсутність системності. У проєкті є чітке розділення відповідальності: браузер відповідає за взаємодію з користувачем, JavaScript керує станом інтерфейсу, сервер приймає запити, а файл бази даних зберігає результат роботи. Завдяки цьому ResourceFlow можна розглядати не як окрему сторінку, а як завершений прототип інформаційної системи [33].

Окрему роль відіграє історія змін, оскільки вона підвищує контрольованість системи. Коли користувач змінює статус ресурсу або працює із заявкою, відповідна дія може бути відображена в журналі. Такий підхід допомагає відстежувати послідовність операцій і краще розуміти, як змінювався стан ресурсів протягом часу.

Таким чином, у межах реалізації серверної частини та структури проєкту було створено основу, яка забезпечує запуск системи, відображення інтерфейсу, обмін даними між браузером і сервером та збереження інформації. Саме ця основа дозволила перейти до реалізації сторінок, форм і користувацьких сценаріїв, описаних у наступних підрозділах.

4.2. Реалізація авторизації, ролей доступу та головної панелі

Одним із перших елементів реалізації стала сторінка авторизації та реєстрації. Вона є початковою точкою взаємодії користувача із системою, тому її потрібно було зробити зрозумілою, без зайвих елементів і з чітким поділом між входом та створенням нового облікового запису. Користувач вводить email і пароль, після чого система перевіряє наявність відповідного запису у базі даних. Під час реалізації авторизації було враховано, що система не повинна пропускати

користувача з неправильними даними. Якщо email або пароль не відповідають збереженому запису, доступ до робочої панелі не надається. Така перевірка є необхідною навіть для навчального прототипу, оскільки вона демонструє базовий принцип контролю входу в інформаційну систему. Це також підвищує довіру до роботи системи з боку користувача.

Реєстрація користувача передбачає створення нового запису в базі даних. На цьому етапі користувач може вказати основні дані, які надалі використовуються у профілі та під час визначення ролі. Роль користувача впливає на те, які дії будуть доступні після входу. Адміністратор має найбільший набір можливостей, менеджер працює з ресурсами та заявками, а звичайний користувач має обмежений доступ.

На рисунку 4.2 наведено сторінку авторизації та реєстрації. Вона показує, як користувач починає роботу із системою, вводить облікові дані та переходить до робочого середовища після успішної перевірки.

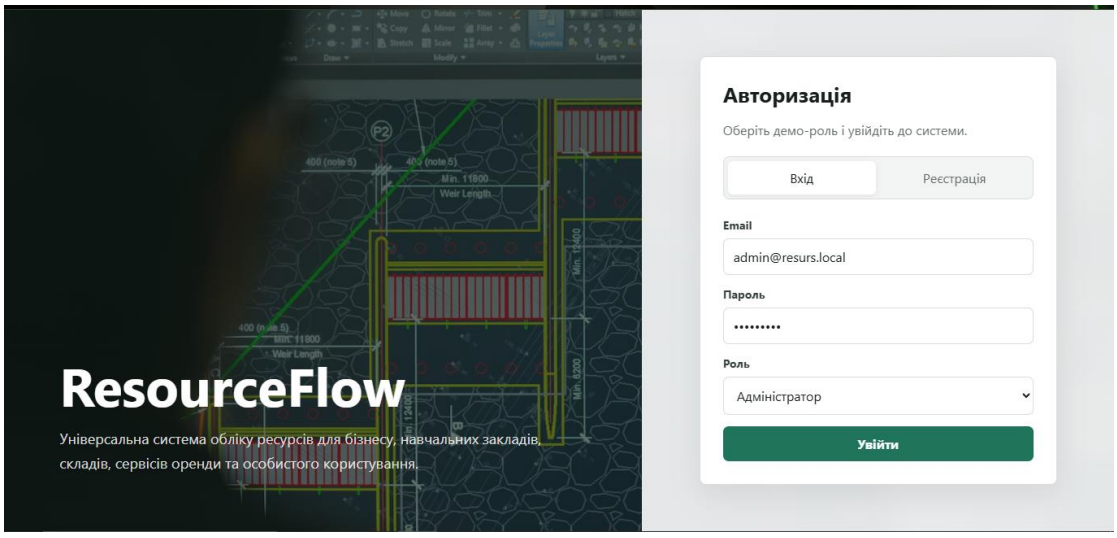


Рисунок 4.2 – Сторінка авторизації та реєстрації користувача

Після успішного входу користувач потрапляє до головної панелі. Її завдання полягає в тому, щоб швидко показати загальний стан системи без необхідності одразу відкривати каталог або звіти. На панелі відображаються

ключові показники: загальна кількість ресурсів, кількість доступних ресурсів, заявки, що очікують розгляду, та ресурси, які перебувають на ремонті.

Головна панель реалізована як робоча область, а не як рекламна сторінка. Вона має допомагати користувачу швидко оцінити ситуацію і перейти до потрібного розділу. Для цього використано картки статистики, таблицю останніх ресурсів і блок історії змін [19]. Така побудова відповідає логіці інформаційної системи, де головне значення має не декоративність, а доступність актуальних даних.

На рисунку 4.3 наведено головну панель ResourceFlow. На ній відображаються статистичні блоки, останні ресурси та історія змін, що дозволяє користувачу швидко оцінити поточний стан системи.

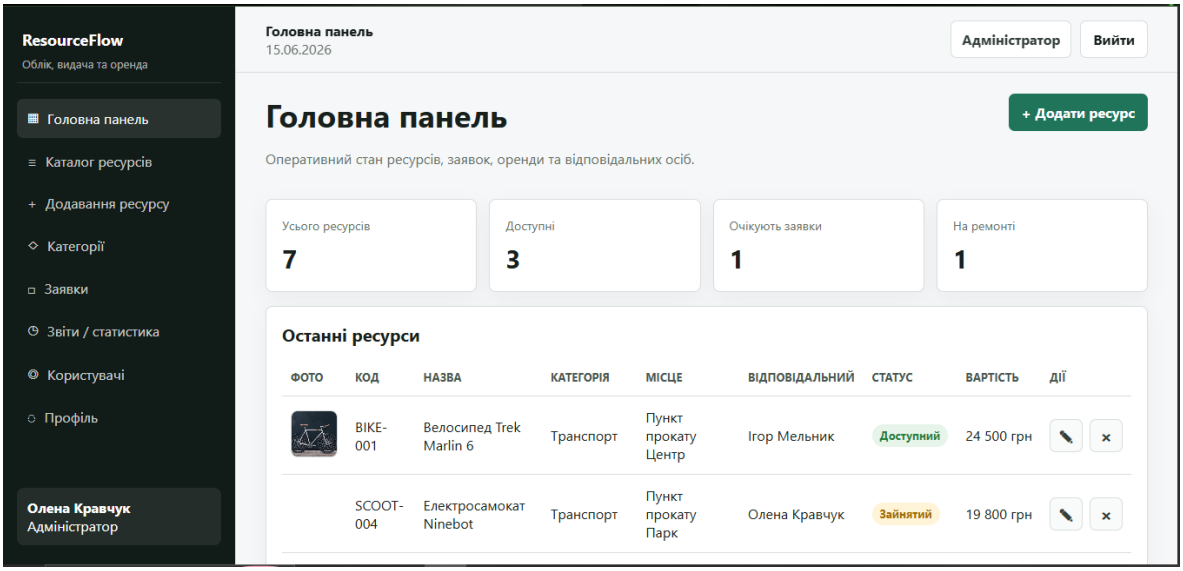


Рисунок 4.3 – Головна панель системи ResourceFlow

Ліве меню забезпечує перехід між основними розділами системи. У ньому розміщені посилання на головну панель, каталог ресурсів, додавання ресурсу, категорії, заявки, звіти, користувачів і профіль. Відображення окремих пунктів може залежати від ролі користувача, що дозволяє не показувати зайві дії тим, хто не має відповідних прав.

Побудова меню має важливе значення для зручності системи. Користувач повинен швидко розуміти, де переглянути ресурси, де створити новий запис, а

де знайти заявки або звіти. Якщо навігація буде складною, система втратить практичну цінність, навіть якщо всі функції технічно реалізовані правильно. Тому меню було зроблено постійним орієнтиром для переходу між основними сторінками.

Під час реалізації інтерфейсу було обрано робочий, а не рекламний формат. У системі немає зайвих промо-блоків або декоративних екранів, тому користувач після входу одразу переходить до дій. Це відповідає призначенню ResourceFlow: система створена не для презентації послуги, а для щоденного обліку ресурсів, заявок і відповідальних осіб.

Рольова модель реалізована через перевірку поточного користувача та його повноважень. Наприклад, адміністратор може керувати ресурсами, категоріями та користувачами, тоді як звичайний користувач переважно переглядає доступні записи та створює заявки. Це робить інтерфейс простішим і зменшує ризик випадкових змін у системі.

Розмежування доступу також допомагає уникнути плутанини під час роботи кількох користувачів. Якщо звичайному користувачу показати кнопки видалення або редагування категорій, він може сприйняти їх як доступні дії. Тому інтерфейс повинен не лише блокувати заборонені операції, а й візуально не перевантажувати користувача зайвими можливостями. Це робить роботу з системою більш зрозумілою для кожної ролі. Крім того, такий підхід зменшує кількість помилкових дій під час щоденного використання [34].

Для адміністратора, навпаки, важливо мати повний набір інструментів керування. Він повинен бачити розділи користувачів, категорій, ресурсів і звітів, оскільки відповідає за наповнення системи та актуальність даних. Менеджер отримує проміжний рівень доступу, достатній для роботи із заявками та поточними статусами ресурсів.

Окрему увагу було приділено профілю користувача. У профілі відображаються персональні дані та роль, з якою користувач працює в системі. Наявність такого розділу робить застосунок більш завершеним, оскільки користувач може бачити, під яким обліковим записом він увійшов і які права має.

На рисунку 4.4 наведено відображення профілю користувача та його ролі в системі. Цей елемент інтерфейсу показує, під яким обліковим записом виконується робота і які повноваження має поточний користувач.

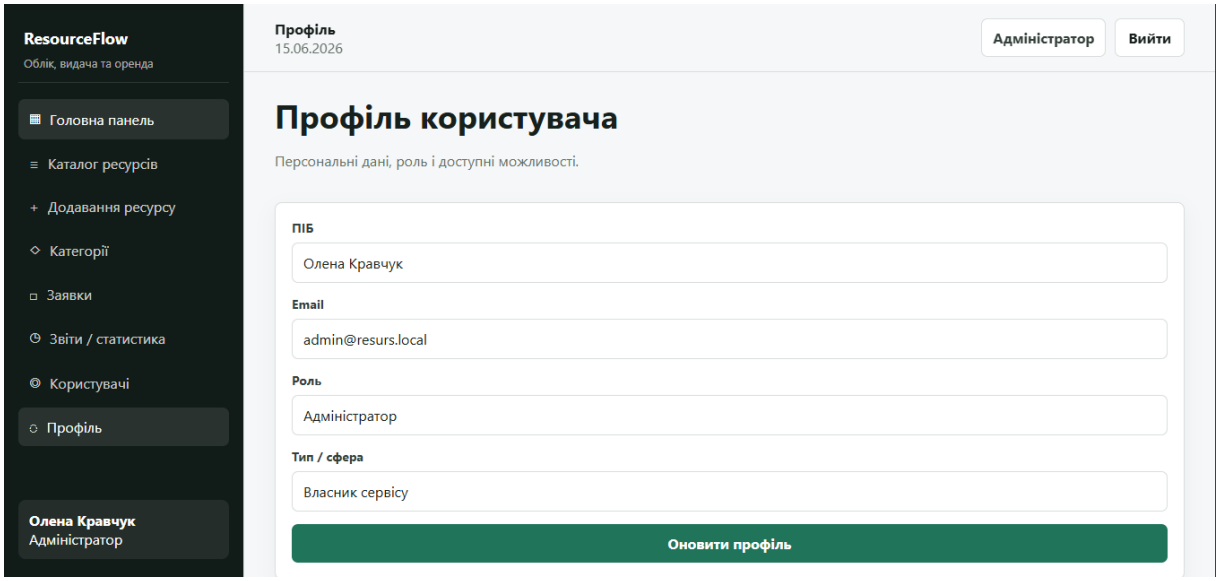


Рисунок 4.4 – Відображення профілю та ролі користувача

Під час реалізації головної панелі було враховано адаптивність. На широкому екрані статистичні блоки розміщуються в кілька колонок, а на меншій ширині вони можуть перебудовуватися так, щоб залишатися читабельними. Це важливо, оскільки система може використовуватися не лише за робочим комп’ютером, а й на ноутбуці або телефоні.

Адаптивність головної панелі має практичне значення для ситуацій, коли облік ведеться не тільки з офісного місця. Наприклад, відповідальна особа може перевіряти ресурс під час видачі або повернення і відкривати систему з мобільного пристрою. У такому випадку картки статистики, меню та кнопки повинні залишатися доступними без зайвого масштабування сторінки.

Блок історії змін було розміщено так, щоб він не створював зайву горизонтальну прокрутку на сторінці. Для браузерної версії важливо, щоб основний контент не виходив за межі екрана, інакше користувачу доводиться постійно рухати сторінку вбік. Тому історію змін доцільно розміщувати внизу або в окремому блоці, який не збільшує ширину всієї сторінки.

Історія змін доповнює головну панель, оскільки показує не тільки поточні показники, а й останні дії в системі. Це допомагає адміністратору швидко помітити, що ресурс був змінений, заявка отримала новий статус або додано новий запис. У невеликій організації такий журнал може частково виконувати роль простого аудиту дій користувачів.

Таким чином, сторінка входу, рольова модель, головна панель і профіль користувача утворюють початковий рівень роботи із системою. Вони визначають, хто саме користується застосунком, які дії йому доступні та яку інформацію він бачить одразу після входу. Без цих елементів каталог ресурсів був би менш контрольованим і не забезпечував би потрібного рівня організації. Це дозволяє підтримувати порядок навіть при роботі кількох користувачів.

Реалізація авторизації, ролей і головної панелі сформувала основу користувацької взаємодії із системою. Після входу користувач одразу отримує доступ до робочого середовища, бачить ключові показники та може перейти до потрібного розділу. Це робить ResourceFlow зручним інструментом для щоденного обліку ресурсів [27].

4.3. Реалізація інтерфейсів користувача та адміністратора

Центральним розділом системи ResourceFlow є каталог ресурсів. Саме в ньому користувач переглядає наявні об’єкти, аналізує їх статус, категорію, відповідальну особу, місце зберігання та вартість. Каталог реалізовано у вигляді таблиці або списку записів, що дозволяє швидко порівнювати ресурси між собою і знаходити потрібний об’єкт. Такий формат подання є зручним для щоденної роботи, оскільки користувач одразу бачить основні характеристики ресурсу без необхідності відкривати додаткові сторінки.

Для підвищення зручності в каталозі реалізовано пошук і фільтрацію. Пошук дозволяє знаходити ресурс за назвою або іншими текстовими даними, а фільтри допомагають обмежити список за категорією чи статусом. Це особливо важливо у випадку, коли кількість ресурсів збільшується і ручний перегляд

усього списку стає незручним. Завдяки цим інструментам користувач може швидко перейти від загального переліку до потрібної групи об’єктів.

На рисунку 4.5 наведено каталог ресурсів із пошуком і фільтрацією. У цьому розділі користувач може переглядати записи, аналізувати статуси, категорії та швидко знаходити потрібний ресурс.

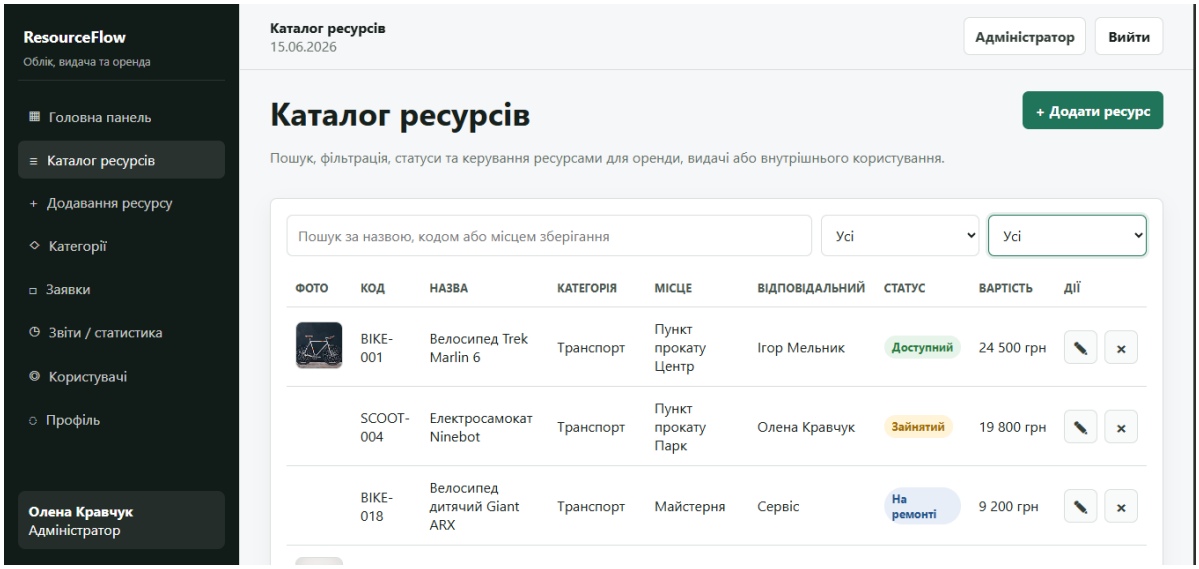


Рисунок 4.5 – Каталог ресурсів із пошуком і фільтрацією

Форма додавання ресурсу реалізована як один із головних інструментів наповнення системи даними. У ній користувач із відповідними правами вводить назву ресурсу, код, категорію, місце зберігання, відповідальну особу, статус, вартість та за потреби додає фотографію. Фото є необов’язковим полем, тому ресурс можна створити навіть тоді, коли зображення ще не підготовлене. Це робить процес внесення даних гнучкішим, оскільки користувач може спочатку зареєструвати ресурс, а зображення додати пізніше.

Під час реалізації форми було важливо зробити її не перевантаженою. Поля згруповані за змістом, щоб користувач послідовно заповнював основну інформацію, облікові дані та додаткові параметри. Такий підхід зменшує кількість помилок і робить процес додавання ресурсу зрозумілішим [15]. Крім того, логічне групування полів допомагає швидше орієнтуватися у формі та не пропускати важливі дані.

На рисунку 4.6 наведено форму додавання та редагування ресурсу. Вона містить основні поля для опису ресурсу, вибору категорії, визначення статусу, відповідальної особи та додавання необов’язкового фото.

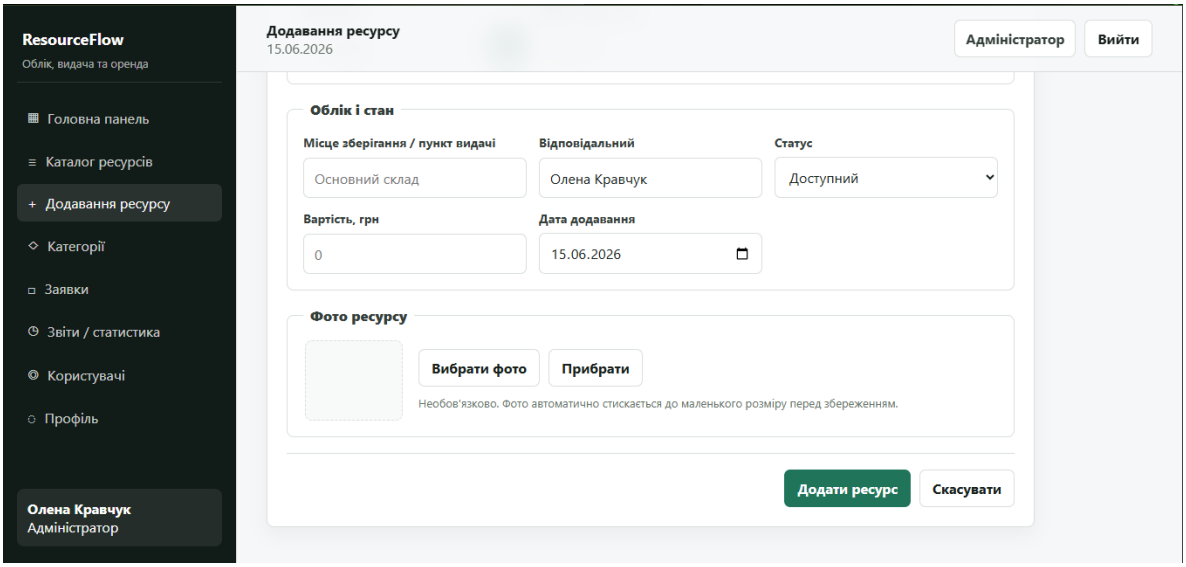


Рисунок 4.6 – Форма додавання та редагування ресурсу

Редагування ресурсу використовує схожу логіку, але відкриває вже існуючі дані. Користувач може змінити назву, категорію, статус, відповідальну особу або інші параметри. Після збереження оновлена інформація відображається у каталозі та може впливати на статистику головної панелі. Це забезпечує актуальність обліку.

Окреме значення має робота зі статусами ресурсів. Статус дозволяє швидко визначити, чи можна використовувати об’єкт, чи він уже зайнятий, списаний або перебуває на ремонті. У практичній роботі це зменшує кількість помилок, тому що користувач не повинен уточнювати стан ресурсу в окремих документах або через усні повідомлення.

Необов’язкове фото ресурсу робить картку більш наочною, але не ускладнює облік. У багатьох випадках під час первинного внесення інформації фотографії може ще не бути, тому вимагати її обов’язково недоцільно. Система дозволяє спочатку створити запис, а зображення додати пізніше під час редагування.

У каталозі фотографії повинні відображатися компактно, щоб вони не займали занадто багато місця і не заважали перегляду текстових даних. Якщо фото відсутнє, інтерфейс не повинен виглядати як помилка. Замість цього достатньо залишити нейтральне місце або просто не відображати велике порожнє зображення, щоб список ресурсів залишався акуратним.

Керування категоріями реалізовано окремим розділом, оскільки категорії є важливою частиною структурування ресурсів. Адміністратор може додавати нові категорії, редагувати їх назви та описи, а також видаляти непотрібні записи. Крім того, можливість додати категорію безпосередньо під час створення ресурсу робить роботу швидшою, бо користувачу не потрібно залишати форму.

Категорії роблять систему універсальною. Якщо користувач веде облік велосипедів, він може створити категорії для транспорту, аксесуарів і ремонтного інструменту. Якщо система використовується в офісі, категорії можуть відповідати ноутбукам, моніторам, меблям або ліцензіям. Завдяки цьому ResourceFlow не обмежується однією сферою застосування [16].

На рисунку 4.7 наведено сторінку керування категоріями. У цьому розділі адміністратор може створювати, редагувати та видаляти категорії, які використовуються для групування ресурсів.

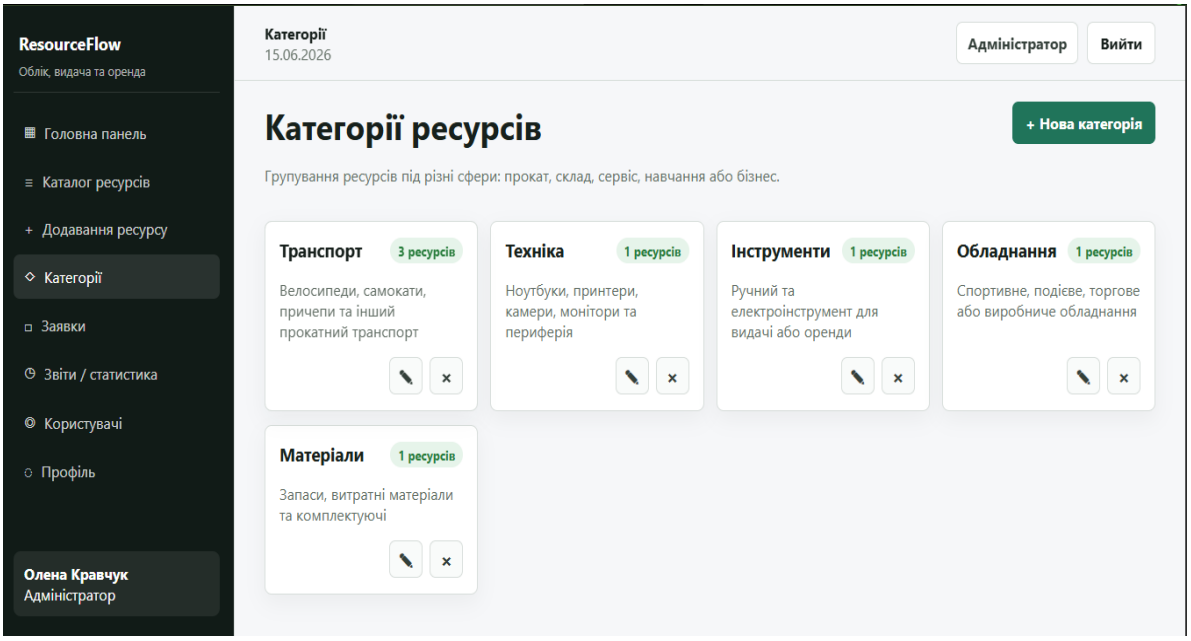


Рисунок 4.7 – Керування категоріями ресурсів

Розділ заявок реалізує процес запиту на використання ресурсу. Користувач створює заявку, обирає потрібний ресурс і вказує мету або опис використання. Далі менеджер або адміністратор може змінити статус заявки, наприклад погодити її, відхилити або залишити в очікуванні. Це дозволяє контролювати не тільки наявність ресурсів, а й процес їх використання. Завдяки такому підходу система фіксує не лише сам ресурс, а й намір користувача отримати його для певної потреби.

Під час зміни статусу заявки важливо зберігати логіку зв’язку між ресурсом і запитом. Якщо ресурс зайнятий або перебуває на ремонті, система повинна відображати це у відповідному статусі, щоб інші користувачі не сприймали його як доступний. Саме тому заявки і статуси ресурсів розглядаються як взаємопов’язані частини системи. Це допомагає уникнути ситуацій, коли один і той самий ресурс одночасно вважається доступним для кількох користувачів.

Заявки допомагають формалізувати процес використання ресурсів. Без такого механізму передача майна часто відбувається неструктуровано: користувач домовляється усно, а відповідальна особа може не зафіксувати факт видачі. Наявність заявки створює запис, який можна переглянути, погодити, відхилити або проаналізувати пізніше. Крім того, заявка може використовуватися як підтвердження того, що користувач справді звертався за конкретним ресурсом.

Для менеджера розділ заявок є робочим інструментом контролю. Він бачить, які ресурси потрібні користувачам, які запити ще очікують відповіді, а які вже були оброблені. У поєднанні з історією змін це дозволяє краще відстежувати рух ресурсів і не втрачати важливі дії. Такий підхід підвищує прозорість роботи системи, оскільки кожна заявка має власний стан і може бути перевірена відповідальною особою.

На рисунку 4.8 наведено сторінку заявок на використання ресурсів. Вона демонструє процес перегляду запитів, контролю їх стану та зміни статусів

менеджером або адміністратором. Через цей розділ забезпечується зв’язок між користувачами, ресурсами та відповідальними особами, що робить процес видачі більш організованим.

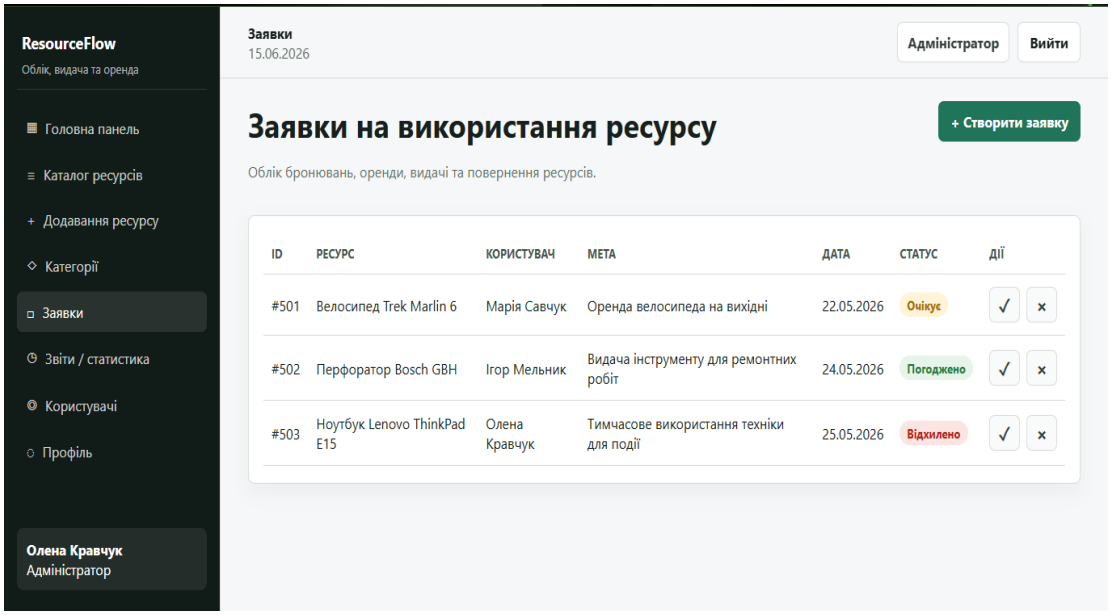


Рисунок 4.8 – Робота із заявками на використання ресурсів

Звіти та статистика реалізовані для швидкого узагальнення даних. Користувач із відповідними правами може переглянути загальну кількість ресурсів, розподіл за статусами, кількість заявок і стан використання майна. Такі дані допомагають швидко оцінити ситуацію без ручного підрахунку записів у каталозі.

Звітність є важливою для адміністратора або керівника, оскільки окремі записи не завжди дають повне уявлення про стан ресурсів. Наприклад, каталог показує конкретні об’єкти, але статистика дозволяє одразу побачити загальну кількість доступних або проблемних позицій. Це прискорює прийняття рішень щодо ремонту, закупівлі або списання.

У майбутньому сторінка звітів може бути розширена експортом у PDF або Excel, але навіть базова статистика вже виконує корисну функцію. Вона показує, що система не тільки зберігає дані, а й перетворює їх на узагальнену інформацію, придатну для аналізу.

На рисунку 4.9 наведено сторінку звітів та статистики. Вона показує, як система узагальнює інформацію про ресурси, заявки та поточний стан обліку у зручному для перегляду вигляді.

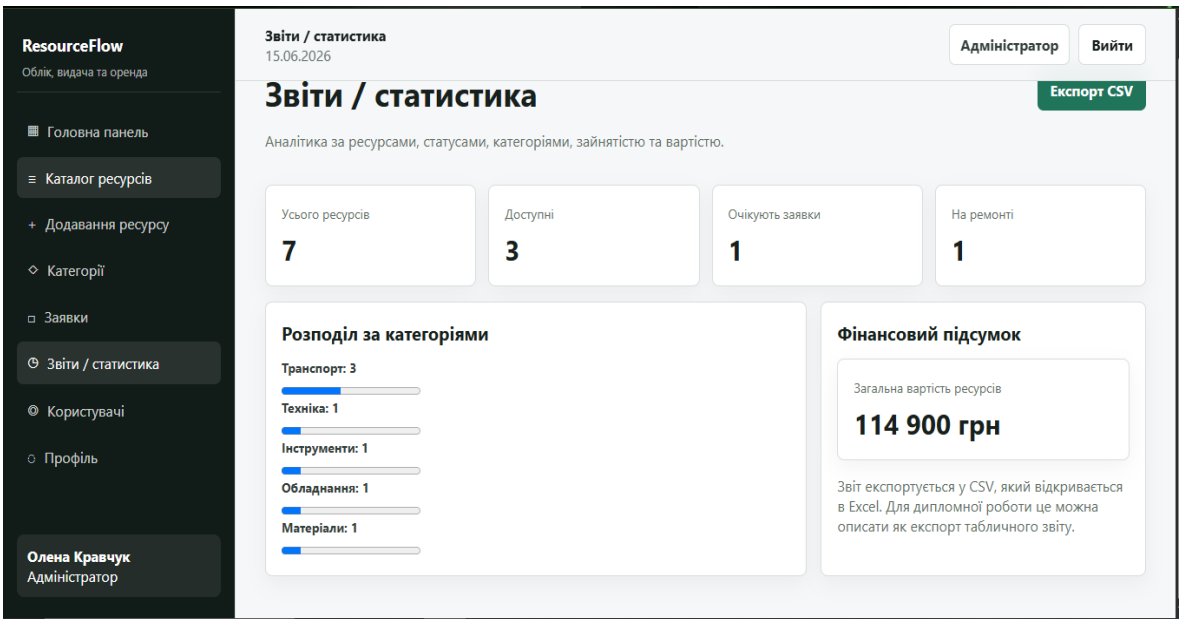


Рисунок 4.9 – Сторінка звітів та статистики системи

Важливою частиною реалізації є адаптивність каталогу, форм, категорій і заявок. На великих екранах таблиці можуть займати більше місця, а на мобільних пристроях окремі блоки повинні перебудовуватися так, щоб користувач міг продовжувати роботу без втрати доступу до кнопок і полів. Це підвищує практичну цінність системи, оскільки облік ресурсів може виконуватися безпосередньо під час перевірки або видачі майна.

Під час адаптації таблиць важливо не допустити ситуації, коли вся сторінка стає надто широкою. Якщо горизонтальна прокрутка потрібна тільки для таблиці, вона повинна обмежуватися межами цього блоку, а не переноситися на весь інтерфейс. Такий підхід робить систему зручнішою у браузері та на мобільних пристроях [13].

Окремо варто зазначити, що всі розділи системи пов’язані між собою через спільні дані. Категорія, створена в одному розділі, використовується у формі ресурсу; ресурс із каталогу може бути обраний у заявці; зміна статусу

впливає на статистику; а історія змін відображає важливі операції. Саме така взаємодія формує цілісність програмної реалізації.

Реалізовані розділи утворюють єдиний робочий цикл. Спочатку адміністратор створює категорії, потім додає ресурси, користувачі переглядають каталог і подають заявки, менеджер змінює статуси, а статистика показує поточний стан системи. Така послідовність демонструє, що ResourceFlow є не набором окремих сторінок, а узгодженою інформаційною системою.

Таким чином, у третьому підрозділі реалізація зосереджена на практичних діях користувача: перегляді ресурсів, заповненні форм, роботі з категоріями, заявками та звітами. Саме ці можливості забезпечують основну цінність системи й дозволяють використовувати її для обліку різних видів ресурсів у реальному середовищі.

4.4. Висновки по розділу

У четвертому розділі було розглянуто практичну реалізацію веб-орієнтованої інформаційної системи ResourceFlow. Описано структуру проєкту, серверну частину, збереження даних у файлі data/db.json, реалізацію авторизації, ролей доступу, головної панелі, каталогу ресурсів, форм додавання і редагування, категорій, заявок, звітів і статистики. Також було показано, як окремі програмні модулі взаємодіють між собою під час виконання основних дій користувача.

У межах реалізації було створено робочий прототип, який дозволяє користувачам виконувати основні операції з ресурсами та заявками в одному веб-середовищі. Система має логічно організований інтерфейс, підтримує рольовий доступ, зберігає дані після виходу та може адаптуватися до різних типів ресурсів. Наведені рисунки демонструють основні сторінки системи та наочно підтверджують результати практичної розробки. Отже, реалізований застосунок відповідає поставленим вимогам і може використовуватися як основа для подальшого вдосконалення.

РОЗДІЛ 5. ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ, СУПРОВІД ТА НАПРЯМИ РОЗВИТКУ ПРОЄКТУ

5.1. Стратегії та методи тестування

Тестування є завершальним етапом перевірки працездатності інформаційної системи, під час якого визначається, чи відповідає розроблений веб-застосунок поставленим вимогам. Для системи ResourceFlow тестування було спрямоване на перевірку основних користувацьких сценаріїв: авторизації, перегляду ресурсів, пошуку, фільтрації, додавання та редагування ресурсів, роботи із заявками, історією змін і статистикою [24].

Стратегія тестування ґрунтувалася на сценарному підході. Для кожної важливої функції було визначено дію користувача, очікуваний результат і фактичний стан виконання. Такий підхід є доцільним для веб-системи обліку ресурсів, оскільки дозволяє перевіряти не лише окремі кнопки або форми, а повний шлях користувача від входу до збереження даних.

Першим напрямом перевірки було функціональне тестування. Воно охоплювало операції, які користувач виконує найчастіше: вхід у систему, відкриття головної панелі, перегляд каталогу, додавання нового ресурсу, редагування існуючого запису, видалення категорії, створення заявки та зміну її статусу. Для кожної дії оцінювалося, чи система правильно реагує на натискання кнопок, чи не зникають введені дані та чи оновлюється інтерфейс після збереження.

Другим напрямом була перевірка коректності даних. У системі ResourceFlow важливо, щоб ресурси, категорії, користувачі та заявки залишалися узгодженими між собою. Наприклад, після створення нової категорії вона повинна з’явитися у формі додавання ресурсу, а після редагування ресурсу оновлена інформація має відображатися у каталозі та на головній панелі. Також перевірялося, чи не створюються порожні записи у випадку незаповнення обов’язкових полів.

Окремо тестувалися негативні сценарії. До них належать спроба входу з неправильним email або паролем, збереження ресурсу без назви, вибір неіснуючої категорії, повторне натискання кнопки збереження та робота з порожнім полем фото. Такі перевірки потрібні для того, щоб система не лише виконувала правильні дії, а й стабільно реагувала на помилки користувача [31].

У таблиці 5.1 наведено приклади тестових сценаріїв, які використовувалися для перевірки основних функцій системи ResourceFlow. Вони охоплюють авторизацію, роботу з ресурсами, заявки, фільтрацію та перевірку серверного API.

Таблиця 5.1

Результати перевірки основних сценаріїв системи

№	Сценарій перевірки	Очікуваний результат	Стан
1	Вхід із правильними даними	Користувач переходить до головної панелі	Пройдено
2	Вхід із неправильним паролем	Система не відкриває панель і повідомляє про помилку	Пройдено
3	Додавання ресурсу без фото	Ресурс зберігається, поле фото залишається порожнім	Пройдено
4	Пошук і фільтрація ресурсу	У каталозі залишаються записи, що відповідають умовам	Пройдено
5	Створення та зміна статусу заявки	Заявка зберігається, а зміна статусу потрапляє до історії	Пройдено
6	Перевірка API /api/db	Сервер повертає коректний JSON із даними системи	Пройдено

Результати, наведені у таблиці 5.1, показують, що основні функції системи виконуються відповідно до очікуваної логіки. Особливу увагу було приділено авторизації, оскільки помилка в цьому модулі може призвести до неправильного доступу до панелі керування. Після виправлення логіки входу система перестала пропускати користувача з неправильними даними, що підвищило достовірність рольової моделі.

Тестування форм додавання та редагування ресурсів проводилося окремо, оскільки ці форми містять найбільшу кількість полів. Перевірялося введення назви, коду, категорії, місця зберігання, відповідальної особи, статусу, вартості

та необов’язкового фото. У результаті було підтверджено, що фото не є обов’язковим полем, а відсутність зображення не блокує створення ресурсу.

Пошук і фільтрація перевірялися на різних наборах ресурсів. Система повинна швидко знаходити записи за назвою, категорією або статусом, не змінюючи самі дані. Такий сценарій важливий для практичної експлуатації, тому що при збільшенні кількості ресурсів користувач не повинен переглядати весь список вручну [32].

Окремо перевірялася робота інтерфейсу на різних розмірах екрана. Для цього аналізувалося відображення головної панелі, каталогу ресурсів, форм додавання і редагування, таблиць, блоку фото та сторінки заявок. Основною вимогою було коректне розміщення елементів без накладання тексту та втрати доступу до кнопок керування.

Під час перевірки адаптивності враховувалося, що користувач може працювати із системою не лише з комп’ютера, а й з ноутбука, планшета або телефону. Тому оцінювалася ширина таблиць, поведінка меню, розташування кнопок, читабельність полів введення та можливість натиснути потрібну дію без горизонтального зміщення всієї сторінки. Це дозволило уникнути ситуації, коли інтерфейс формально відкривається на телефоні, але фактично ним незручно користуватися. Особливу увагу було приділено зручності роботи з формами на малих екранах.

Важливим елементом тестування була перевірка збереження даних після виходу із системи або перезапуску сервера. Оскільки система використовує файл data/db.json як окреме сховище, необхідно було переконатися, що додані ресурси, користувачі, категорії та заявки не зникають після оновлення сторінки. Цей сценарій є принциповим, тому що саме збереження інформації відрізняє систему від простого статичного макета.

На рисунку 5.1 наведено загальну послідовність тестування системи ResourceFlow. Схема показує, що перевірка охоплювала планування сценаріїв, функціональне тестування, оцінку адаптивності, перевірку API та повторну перевірку після усунення помилок.

Рисунок 5.1 – Послідовність тестування системи ResourceFlow

Послідовність, показана на рисунку 5.1, відображає циклічний характер тестування. Якщо під час перевірки виявлялася помилка, вона виправлялася у коді, після чого відповідний сценарій проходив повторну перевірку. Такий підхід дозволяє не обмежуватися одноразовим запуском системи, а поступово підвищувати якість програмної реалізації.

Окремо перевірялася серверна частина. Для цього аналізувалася відповідь маршруту /api/db, структура отриманого JSON та можливість запису оновлених даних у файл [22]. Така перевірка підтверджує, що браузерна частина не працює ізольовано, а взаємодіє із сервером, який відповідає за збереження інформації між сеансами.

Під час тестування також враховувалася зрозумілість інтерфейсу. Система повинна давати користувачу очевидний шлях дій: де додати ресурс, де змінити статус, де знайти заявку, де переглянути статистику. Якщо елемент керування виглядав незручно або займав надто багато місця, його доцільно було переробити, оскільки для інформаційної системи важлива не тільки працездатність, а й щоденна зручність.

Таким чином, тестування ResourceFlow поєднувало перевірку функцій, даних, інтерфейсу, адаптивності та серверної взаємодії. Це дозволило підтвердити, що система відповідає основним вимогам, сформованим у

попередніх розділах, і може використовуватися як робочий прототип веб-орієнтованої інформаційної системи обліку ресурсів.

За результатами тестування встановлено, що основні сценарії роботи системи виконуються коректно. Система дозволяє виконувати вхід, зберігати ресурси, працювати із заявками, переглядати статистику та отримувати дані через серверний маршрут /api/db. Виявлені під час розробки недоліки були усунені до формування фінальної версії проєкту.

5.2. Впровадження, release-перевірки та експлуатація системи

Впровадження системи ResourceFlow передбачає перенесення файлів проєкту в середовище, де веб-застосунок може бути доступний користувачам через браузер. У межах роботи систему було підготовлено до локального запуску та розгортання на VPS-сервері. Такий підхід дозволяє продемонструвати не лише роботу інтерфейсу на комп’ютері розробника, а й практичну можливість використання системи як веб-сервісу.

Локальне впровадження використовується на етапі розробки та первинної перевірки. У цьому випадку розробник запускає сервер на власному комп’ютері, відкриває систему в браузері та перевіряє зміни одразу після редагування файлів. Такий спосіб є зручним для налагодження інтерфейсу, перевірки форм, виправлення стилів і швидкого тестування логіки без перенесення файлів на віддалений сервер [15].

Серверне впровадження передбачає розміщення проєкту на VPS. У такому середовищі система може працювати постійно, а користувачі отримують доступ до неї через IP-адресу або домен. Для ResourceFlow важливо, щоб проєкт не конфліктував з іншими веб-застосунками на сервері, тому під час розгортання необхідно використовувати окрему папку, окремий порт і власну службу запуску.

Файлова структура проєкту спрощує впровадження. Клієнтські файли розміщені окремо від серверної логіки та бази даних, тому адміністратор може

швидко перевірити, де знаходиться інтерфейс, де зберігається серверний код і де розміщено файл data/db.json. Такий поділ є корисним під час перенесення на сервер, резервного копіювання та подальшого супроводу.

Перед випуском оновленої версії доцільно виконувати release-перевірки. До них належать перевірка запуску сервера, доступності сторінки index.html, коректного завантаження CSS і JavaScript, відповіді маршруту /api/db, збереження даних у data/db.json та відсутності критичних помилок у браузері. Такі дії дозволяють зменшити ризик перенесення непрацездатної версії на сервер.

Release-перевірки доцільно виконувати перед кожною зміною, яка впливає на користувацький інтерфейс або збереження даних. Наприклад, після зміни форми додавання ресурсу потрібно перевірити не лише зовнішній вигляд форми, а й створення запису в базі даних. Після зміни авторизації необхідно перевірити як правильний, так і неправильний вхід, щоб уникнути помилкового доступу.

Окремим етапом є перевірка сумісності з уже наявними даними. Якщо у файлі data/db.json вже є ресурси, користувачі та заявки, нова версія системи не повинна пошкодити ці записи. Тому перед впровадженням бажано створити копію файлу бази даних, запустити оновлену версію і переконатися, що старі записи відкриваються без помилок.

Під час розгортання на сервері важливо налаштувати стабільний запуск застосунку. Для цього може використовуватися системна служба, яка автоматично запускає Node.js-сервер після перезавантаження VPS. Такий підхід зручніший, ніж ручний запуск через консоль, тому що система залишається доступною навіть після технічного перезапуску сервера.

Експлуатація системи передбачає регулярну роботу користувачів із каталогом ресурсів, заявками, категоріями та звітами. Адміністратор відповідає за підтримку довідників і користувачів, менеджер контролює заявки та статуси ресурсів, а звичайний користувач переглядає доступні ресурси й створює заявки. Такий розподіл підтримує впорядковану роботу системи після впровадження.

Після впровадження важливо визначити правила користування системою. Наприклад, адміністратор повинен вчасно додавати нові категорії, менеджер має оновлювати статуси заявок, а користувачі повинні створювати заявки тільки на актуальні ресурси. Такі організаційні правила не є частиною програмного коду, але вони впливають на якість даних і практичну користь системи.

У процесі експлуатації особливе значення має актуальність статусів ресурсів. Якщо ресурс фактично зайнятий, але в системі залишається доступним, користувач може подати неправильну заявку. Тому менеджер або адміністратор має регулярно оновлювати статуси: доступний, зайнятий, списаний або на ремонті. Це забезпечує відповідність електронного обліку реальному стану майна.

На рисунку 5.2 наведено загальну схему впровадження та супроводу системи ResourceFlow. Вона показує послідовність оновлення файлів, виконання release-перевірок, запуску сервісу, доступу через браузер, моніторингу та резервного копіювання.

Рисунок 5.2 – Схема впровадження та супроводу системи ResourceFlow

Схема на рисунку 5.2 показує, що впровадження не завершується простим копіюванням файлів. Після перенесення необхідно перевірити запуск,

доступність системи, роботу API, збереження даних і стан служби. Лише після цього систему можна вважати готовою до використання користувачами.

У разі появи помилки після оновлення важливо мати можливість швидко повернути попередню працездатну версію. Для цього перед release-перевіркою бажано зберігати копію проєкту або принаймні копію файлу бази даних. Такий підхід зменшує ризик втрати інформації та скорочує час відновлення роботи системи.

Експлуатаційна документація для такого проєкту може бути короткою, але вона повинна містити основні команди запуску, адресу доступу, опис розташування файлу бази даних, порядок резервного копіювання і перелік типових перевірок. Це дозволяє іншому користувачу або адміністратору швидко зрозуміти, як підтримувати систему без постійного звернення до розробника [9]. Наявність такої інструкції також спрощує передачу проєкту на супровід.

Практична експлуатація системи можлива за умови підтримки актуального стану бази даних і контролю працездатності серверної частини. Оскільки ResourceFlow використовує просту структуру файлів і не має складних зовнішніх залежностей, його можна швидко переносити між середовищами або відновлювати після оновлення.

Отже, впровадження ResourceFlow включає локальну перевірку, серверне розгортання, release-контроль, запуск служби, перевірку доступності та подальший супровід. Такий порядок робіт робить систему придатною не лише для демонстрації в межах бакалаврської роботи, а й для обмеженого практичного використання у невеликій організації або пункті прокату.

5.3. Захист даних, резервне копіювання та моніторинг

Захист даних є важливою складовою супроводу інформаційної системи, навіть якщо вона розглядається як навчальний прототип. У ResourceFlow зберігаються користувачі, ресурси, заявки, категорії та історія змін, тому втрата або пошкодження файлу бази даних може вплинути на роботу всієї системи.

Саме тому необхідно передбачити базові організаційні та технічні заходи безпеки.

Основними об’єктами захисту є облікові записи користувачів, ресурсні записи, заявки та історія змін. Ці дані мають різну цінність, але разом формують цілісну картину обліку. Якщо буде втрачено тільки каталог ресурсів, система перестане виконувати свою основну функцію. Якщо буде пошкоджено історію змін, адміністратору стане важче відстежити, хто і коли змінював записи.

Для навчального проєкту достатньо реалізувати базові принципи безпеки, але важливо розуміти їх подальший розвиток. У промисловій версії паролі не повинні зберігатися у відкритому вигляді, а перевірка доступу має виконуватися не лише на клієнтському рівні, а й на сервері. Це необхідно для того, щоб користувач не міг обійти обмеження шляхом прямого редагування запиту або JavaScript-коду.

Першим напрямом захисту є обмеження доступу до функцій системи залежно від ролі користувача. Адміністратор має розширені права, менеджер працює з ресурсами та заявками, а звичайний користувач має обмежений доступ. Такий поділ зменшує ймовірність випадкового видалення або зміни важливих даних користувачем, який не повинен виконувати адміністративні операції.

Рольова модель також впливає на зручність роботи. Користувач бачить тільки ті дії, які мають сенс для його ролі, тому інтерфейс стає простішим і менш перевантаженим. З погляду захисту це зменшує кількість випадкових помилок, а з погляду ергономіки допомагає швидше виконувати типові операції [24].

Другим напрямом є резервне копіювання. Оскільки дані системи зберігаються у файлі data/db.json, доцільно регулярно створювати копії цього файлу. Резервна копія може формуватися вручну після важливих змін або автоматично за розкладом. Це дозволить відновити стан системи у разі помилкового редагування, збою сервера або пошкодження файлу.

Резервне копіювання має виконуватися не лише формально, а й з перевіркою можливості відновлення. Якщо копія створюється, але жодного разу не перевіряється, адміністратор не може бути впевнений, що вона справді

придатна для використання. Тому періодично потрібно відкривати копію, перевіряти структуру JSON і переконуватися, що система може запуститися з відновленим файлом.

Для зменшення ризику втрати даних доцільно зберігати кілька копій за різні дати. Наприклад, одна копія може створюватися перед оновленням системи, інша після завершення робочого дня, а третя зберігатися як архів за тиждень. Такий підхід дозволяє повернутися не тільки до останнього стану, а й до попередньої стабільної версії даних.

Моніторинг у межах такого проєкту може включати перевірку доступності веб-сторінки, стану серверного процесу, відповіді API та цілісності файлу бази даних. Для VPS-сервера корисним є перегляд статусу служби, журналів запуску та помилок. Навіть базовий моніторинг допомагає швидше виявляти проблеми і скорочувати час простою системи.

Моніторинг також може бути організований на рівні простих регламентних перевірок. Адміністратор може щодня відкривати головну сторінку, перевіряти авторизацію, створювати тестову заявку або переглядати відповідь API. Для невеликої системи цього достатньо, щоб вчасно помітити недоступність сервера, помилки в інтерфейсі або некоректне збереження даних. Такі перевірки можна виконувати вручну без складних інструментів.

Журналювання помилок є перспективним напрямом розвитку супроводу. У поточній реалізації основну інформацію про помилки можна отримати через консоль браузера або журнал серверного процесу. У майбутньому доцільно додати окремий файл логів, у якому будуть фіксуватися помилки запуску, некоректні запити та проблеми запису у файл бази даних. Такі заходи дозволяють підтримувати працездатність системи навіть у випадку технічних збоїв або помилкових дій користувачів.

У таблиці 5.2 наведено основні ризики розвитку й експлуатації системи ResourceFlow, а також можливі напрями їх усунення. Така таблиця дозволяє оцінити слабкі місця прототипу та визначити практичні кроки для його покращення.

Таблиця 5.2

Ризики експлуатації та напрями розвитку системи

№	Ризик або обмеження	Можливий вплив	Напрям усунення
1	Файлова база даних	Обмеження при великій кількості одночасних записів	Перехід на SQLite або PostgreSQL
2	Демонстраційна авторизація	Недостатній рівень захисту для промислового використання	Хешування паролів і серверні сесії
3	Відсутність автоматичного backup	Ризик втрати даних під час збою	Регулярне резервне копіювання data/db.json
4	Обмежена аналітика	Недостатньо даних для глибокого управлінського аналізу	Додавання розширених звітів і графіків
5	Використання порту без домену	Менша зручність доступу для користувачів	Налаштування домену, HTTPS і reverse proxy

Наведені у таблиці 5.2 ризики не означають, що система є непридатною для використання. Вони показують межі поточної реалізації та допомагають визначити, які зміни потрібні для переходу від навчального прототипу до більш надійного програмного продукту. Це особливо важливо для бакалаврської роботи, оскільки демонструє розуміння не лише реалізації, а й подальшої експлуатації системи. Аналіз таких ризиків дозволяє заздалегідь визначити слабкі місця проєкту і спланувати напрями його поступового вдосконалення.

Захист даних також пов’язаний із відповідальністю користувачів. Навіть найкращі технічні механізми не гарантують якісний облік, якщо користувачі вводять неправильні назви, не оновлюють статуси або видаляють потрібні категорії. Тому адміністратор повинен контролювати правила ведення довідників і періодично перевіряти актуальність записів. Такий контроль допомагає підтримувати порядок у системі та зменшує ймовірність накопичення помилкових або застарілих даних.

Наведені заходи не перетворюють навчальний прототип на повністю промислову систему, однак формують базове розуміння супроводу веб-застосунку. Для подальшого використання ResourceFlow у реальних умовах необхідно посилити авторизацію, додати автоматичне резервне копіювання, журналювання помилок та контроль доступу на серверному рівні. Такі кроки

дадуть змогу підвищити надійність системи та краще підготувати її до роботи з більшим обсягом даних.

Отже, захист, резервне копіювання та моніторинг є не додатковими, а необхідними складовими життєвого циклу системи ResourceFlow. Вони забезпечують збереження інформації, контроль доступу, стабільність роботи і можливість швидкого відновлення після помилок або технічних збоїв. Саме ці складові дозволяють розглядати систему не лише як розроблений прототип, а як основу для подальшого практичного впровадження.

5.4. Масштабування, практична доцільність і ризики розвитку

Масштабування ResourceFlow може здійснюватися поступово. На початковому етапі система використовує JSON-файл як просте сховище даних, що є достатнім для демонстраційного або невеликого робочого середовища. Якщо кількість ресурсів і користувачів зростатиме, доцільно перейти на повноцінну систему керування базами даних, наприклад SQLite або PostgreSQL.

Поступове масштабування є доцільним, тому що система вже має логічно відокремлені частини: інтерфейс, серверну обробку та файл даних. Це означає, що в майбутньому можна замінити тільки сховище, залишивши основну структуру інтерфейсу. Наприклад, замість читання і запису data/db.json сервер може працювати з таблицями бази даних, не змінюючи загальну логіку користувацьких сценаріїв [30].

Першим напрямом технічного масштабування є оптимізація роботи з даними. При невеликій кількості ресурсів JSON-файл є зручним і наочним, але зі збільшенням кількості записів зростає потреба у швидкому пошуку, індексах, транзакціях і захисті від одночасного запису. Саме ці задачі краще вирішуються за допомогою повноцінної бази даних.

Другим напрямом є розширення серверного API. У поточній реалізації маршрут /api/db працює як універсальна точка читання і запису всієї бази. Для більшого проєкту доцільно створити окремі маршрути для ресурсів,

користувачів, категорій, заявок і звітів. Це зробить серверну частину більш структурованою та спростить контроль прав доступу.

Практична доцільність системи полягає в її універсальності. ResourceFlow не прив’язана до конкретної галузі, тому може застосовуватися для обліку техніки, інструментів, меблів, транспорту, матеріалів або інвентарю. Адміністратор може змінювати категорії відповідно до потреб організації, а користувачі можуть створювати заявки на використання ресурсів.

Універсальність є однією з головних переваг ResourceFlow. Система може використовуватися у пункті прокату велосипедів, у навчальному закладі для обліку техніки, в офісі для контролю меблів і обладнання, на складі для матеріалів або в майстерні для інструментів. Для цього не потрібно змінювати програмний код, достатньо налаштувати категорії та наповнити каталог відповідними ресурсами.

Практична користь системи зростає тоді, коли в організації є потреба не просто зберігати список майна, а контролювати його стан і використання. Статуси ресурсів дозволяють бачити, що доступне, що зайняте, що списане, а що перебуває на ремонті. Заявки допомагають фіксувати потребу користувачів, а історія змін створює основу для контролю дій.

До напрямів розвитку можна віднести додавання календаря бронювання, строків повернення ресурсів, розширеної аналітики, експорту у PDF або XLSX, повідомлень користувачів, імпорту ресурсів із CSV та повноцінного журналу аудиту. Такі функції дозволять перетворити прототип на більш комплексну систему управління ресурсами. Окремим перспективним напрямом є додавання модуля термінів користування. Для прокату або внутрішнього обліку важливо знати не лише факт видачі ресурсу, а й дату повернення. У майбутньому система може автоматично показувати прострочені ресурси, нагадувати менеджеру про повернення та формувати список активних видач.

Ще одним напрямом розвитку є покращення звітності. Поточна статистика показує базові показники, однак у майбутньому можна додати діаграми за категоріями, статусами, відповідальними особами та періодами. Це дозволить

керівнику швидше оцінювати завантаженість ресурсів і приймати рішення щодо закупівлі, ремонту або списання майна.

Також доцільно розглянути інтеграцію з експортом даних. Для багатьох організацій важливо отримувати таблиці у форматі Excel або звіти у форматі PDF. Такий функціонал зробить систему кориснішою для документообігу, оскільки результати обліку можна буде додавати до звітів, передавати керівництву або зберігати як архів. Серед ризиків розвитку варто відзначити ускладнення інтерфейсу, збільшення навантаження на сервер, потребу у надійнішому захисті облікових записів і необхідність кращої структури бази даних. Тому подальше розширення системи повинно виконуватися поступово, із перевіркою кожного нового модуля та збереженням простоти основних сценаріїв.

Найбільшим ризиком під час розвитку є надмірне ускладнення системи. Якщо додавати багато функцій без узгодженої структури, інтерфейс може стати перевантаженим, а користувачі втратять швидкий доступ до основних дій. Тому кожне нове розширення повинно оцінюватися з погляду практичної користі: чи справді воно допомагає вести облік ресурсів, чи лише збільшує кількість елементів на екрані.

Загалом ResourceFlow має практичну цінність як основа для подальшого розвитку. Система вже демонструє ключові можливості інформаційної системи обліку ресурсів, а її архітектура дозволяє поступово додавати нові функції без повного переписування проєкту [28].

Отже, напрями розвитку ResourceFlow пов’язані з переходом від навчального прототипу до більш повної системи управління ресурсами. Поточна реалізація виконує основні задачі обліку, а подальше масштабування може охоплювати базу даних, безпеку, звіти, календар бронювання, автоматичні нагадування та інтеграцію з документами. Окремо варто врахувати можливість розширення системи за рахунок інтеграції з іншими сервісами. Наприклад, у майбутньому ResourceFlow може підтримувати імпорт ресурсів із таблиць, автоматичне формування звітів або надсилання сповіщень користувачам про

зміну статусу заявки. Такі можливості підвищили б зручність системи та зробили її кориснішою для реального використання.

Додатково варто зазначити, що розвиток ResourceFlow може відбуватися не тільки через додавання нових функцій, а й через покращення якості вже реалізованих можливостей. Наприклад, можна вдосконалити перевірку введених даних у формах, зробити більш детальну історію змін, додати підтвердження важливих дій перед видаленням записів і розширити повідомлення для користувача. Такі покращення не змінюють основну ідею системи, але роблять її більш надійною, зрозумілою та зручною для щоденного використання.

5.5. Висновки по розділу

У п’ятому розділі було розглянуто тестування, впровадження, супровід і напрями розвитку системи ResourceFlow. Було визначено основні тестові сценарії, перевірено роботу авторизації, ресурсів, заявок, фільтрів, API, збереження даних і адаптивного інтерфейсу. Результати тестування підтвердили, що основні функції системи працюють коректно та відповідають поставленим вимогам.

Також було описано особливості впровадження системи, release-перевірки, експлуатацію, захист даних, резервне копіювання та моніторинг. Окремо визначено ризики подальшого розвитку і напрями масштабування, зокрема перехід на повноцінну базу даних, посилення авторизації, автоматизацію резервного копіювання та розширення аналітики. Отже, ResourceFlow можна розглядати як завершений функціональний прототип, який має практичну цінність і може бути поступово розширений у майбутньому.

Крім того, у розділі показано, що система придатна не лише для демонстрації, а й для практичного використання в невеликій організації або пункті прокату. Завдяки простій архітектурі, ролям доступу та збереженню даних ResourceFlow може бути основою для подальшого вдосконалення і впровадження в реальні робочі процеси.

ВИСНОВКИ

У бакалаврській роботі виконано розроблення веб-орієнтованої інформаційної системи обліку ресурсів ResourceFlow. Досягнуто поставленої мети: створено прототип системи, який дозволяє вести облік різних типів ресурсів, зокрема техніки, обладнання, інструментів, меблів, матеріалів, транспорту та інших об’єктів, що можуть використовуватися в організації або пункті прокату.

У процесі виконання роботи проаналізовано предметну область обліку та управління ресурсами, визначено основні проблеми ручного ведення обліку, втрати актуальної інформації, складності контролю статусів і відсутності єдиного середовища для роботи із заявками. Було сформовано ролі користувачів системи: адміністратора, менеджера та звичайного користувача. Для кожної ролі визначено доступні функції, що дозволило побудувати логічну модель взаємодії користувачів із системою.

У роботі сформульовано функціональні та нефункціональні вимоги до системи ResourceFlow. До основних функціональних можливостей належать авторизація та реєстрація користувачів, облік ресурсів, додавання, редагування і видалення записів, керування категоріями, пошук і фільтрація, створення заявок, зміна статусів, перегляд історії змін, статистика та збереження даних. Нефункціональні вимоги охоплюють зручність інтерфейсу, адаптивність, стабільність роботи, простоту розгортання та можливість подальшого розвитку системи.

Спроєктовано архітектуру веб-застосунку, яка складається з клієнтської частини, серверного HTTP API та файлової бази даних. Клієнтська частина реалізована за допомогою HTML, CSS і JavaScript, серверна частина створена на Node.js, а дані зберігаються у файлі data/db.json. Така архітектура є зрозумілою, достатньою для навчального проєкту та дозволяє продемонструвати повний цикл роботи системи: від дії користувача в браузері до збереження інформації на сервері.

Практична реалізація системи охоплює основні сторінки веб-застосунку: авторизацію, головну панель, каталог ресурсів, додавання та редагування ресурсу, категорії, заявки, звіти, користувачів і профіль. Особливу увагу приділено формам додавання та редагування ресурсів, оскільки саме вони забезпечують наповнення системи даними. Також реалізовано можливість додавання необов'язкового фото ресурсу, що робить картки ресурсів більш інформативними, але не ускладнює процес заповнення.

Результати тестування підтвердили працездатність основних сценаріїв: вхід користувача, перевірку неправильних даних авторизації, додавання ресурсу, редагування запису, створення категорії, пошук і фільтрацію, створення заявки, зміну статусу та збереження інформації після перезапуску системи. Окремо перевірено адаптивність інтерфейсу, щоб система могла використовуватися не лише на комп'ютері, а й на пристроях із меншою шириною екрана.

Практичне значення роботи полягає в тому, що ResourceFlow не прив'язана до однієї конкретної галузі. Систему можна застосовувати для обліку ресурсів у навчальному закладі, офісі, складі, майстерні, пункті прокату велосипедів, сервісі оренди обладнання або іншій організації, де потрібно контролювати наявність, стан і використання майна.

Окремо розглянуто питання впровадження, release-перевірок, експлуатації, резервного копіювання, моніторингу та напрямів розвитку системи. Це дозволяє оцінити ResourceFlow не лише як набір сторінок і форм, а як приклад прикладної інформаційної системи, яку можна запускати, підтримувати, оновлювати та поступово розширювати після впровадження.

Подальший розвиток проєкту може включати перехід від JSON-файлу до повноцінної бази даних, посилення серверної авторизації, хешування паролів, додавання календаря бронювання, термінів повернення ресурсів, автоматичних повідомлень, розширеної аналітики, експорту звітів у PDF або Excel. У межах бакалаврської роботи реалізовано завершений функціональний прототип, який демонструє застосування сучасних веб-технологій для розв'язання практичної задачі обліку та управління ресурсами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ: ДП «УкрНДНЦ», 2016 (дата звернення: 10.03.2026).

2) MDN Web Docs. HTML: HyperText Markup Language. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення: 15.03.2026).

3) MDN Web Docs. CSS: Cascading Style Sheets. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 16.03.2026).

4) MDN Web Docs. JavaScript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 16.03.2026).

5) MDN Web Docs. Fetch API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 25.03.2026).

6) MDN Web Docs. File API: Using files from web applications. URL: https://developer.mozilla.org/en-US/docs/Web/API/File_API/Using_files_from_web_applications (дата звернення: 25.03.2026).

7) WHATWG. HTML Living Standard. URL: https://html.spec.whatwg.org/multipage/ (дата звернення: 26.03.2026).

8) W3C. Media Queries Level 4. URL: https://www.w3.org/TR/mediaqueries-4/ (дата звернення: 26.03.2026).

9) W3C. Web Content Accessibility Guidelines (WCAG) 2.2. URL: https://www.w3.org/TR/WCAG22/ (дата звернення: 27.03.2026).

10) Node.js Documentation. About Node.js. URL: https://nodejs.org/en/about (дата звернення: 27.03.2026).

11) Node.js Documentation. HTTP module. URL: https://nodejs.org/api/http.html (дата звернення: 01.04.2026).

12) Node.js Documentation. File system module. URL: https://nodejs.org/api/fs.html (дата звернення: 01.04.2026).

13) npm Docs. package.json. URL: https://docs.npmjs.com/cli/v10/configuring-npm/package-json (дата звернення: 01.04.2026).

14) JSON.org. Introducing JSON. URL: https://www.json.org/json-en.html (дата звернення: 15.04.2026).

15) Bray T. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259. URL: https://www.rfc-editor.org/rfc/rfc8259 (дата звернення: 15.04.2026).

16) Fielding R. T., Reschke J. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230. URL: https://www.rfc-editor.org/rfc/rfc7230 (дата звернення: 15.04.2026).

17) OWASP Cheat Sheet Series. Authentication Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html (дата звернення: 16.04.2026).

18) OWASP Cheat Sheet Series. Authorization Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html (дата звернення: 18.04.2026).

19) OWASP Cheat Sheet Series. Input Validation Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html (дата звернення: 19.04.2026).

20) OWASP Cheat Sheet Series. Logging Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html (дата звернення: 25.04.2026).

21) OWASP Foundation. OWASP Top Ten. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 25.04.2026).

22) Microsoft. REST API Guidelines. URL: https://github.com/microsoft/api-guidelines (дата звернення: 26.04.2026).

23) Microsoft Learn. Web application architecture. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/web-queue-worker (дата звернення: 27.04.2026).

24) Microsoft Learn. Static web app architecture approaches. URL: https://learn.microsoft.com/en-us/azure/architecture/web-apps/ (дата звернення: 28.04.2026).

25) Nielsen J. 10 Usability Heuristics for User Interface Design. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 01.05.2026).

26) Google web.dev. Learn Responsive Design. URL: https://web.dev/learn/design/ (дата звернення: 02.05.2026).

27) Google web.dev. Learn Forms. URL: https://web.dev/learn/forms/ (дата звернення: 03.05.2026).

28) SQLite Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 04.05.2026).

29) PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 10.05.2026).

30) Git Documentation. URL: https://git-scm.com/doc (дата звернення: 15.05.2026).

31) GitHub Docs. About READMEs. URL: https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-readmes (дата звернення: 15.05.2026).

32) ISO/IEC 25010. Systems and software Quality Requirements and Evaluation. URL: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010 (дата звернення: 16.05.2026).

33) Sommerville I. Software Engineering. 10th edition. Pearson, 2015. 816 (дата звернення: 16.05.2026).

34) Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th edition. McGraw-Hill Education, 2019. 672(дата звернення: 16.05.2026).

35) Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 560(дата звернення: 16.05.2026).

36) Freeman E., Robson E. Head First JavaScript Programming. O'Reilly Media, 2014. 704 (дата звернення: 16.05.2026).

ДОДАТКИ

ДОДАТОК А

Демонстрація інтерфейсу системи

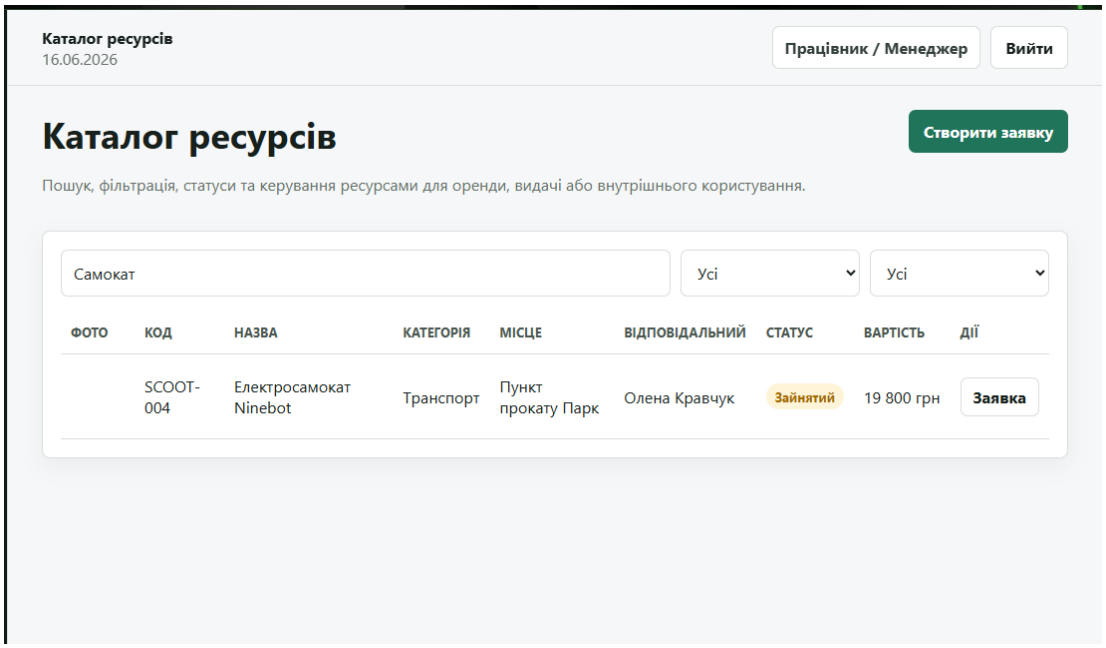


Рисунок А.1 –Фільтрація ресурсів за категорією

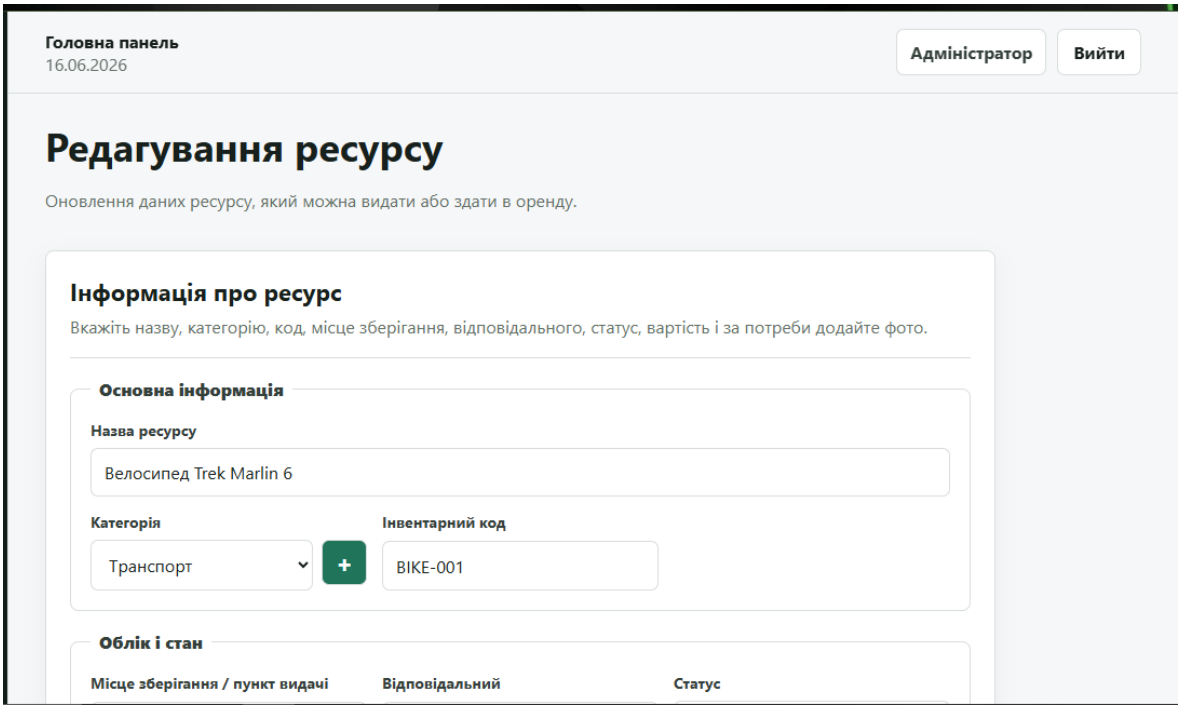


Рисунок А.2 –Редагування даних ресурсу

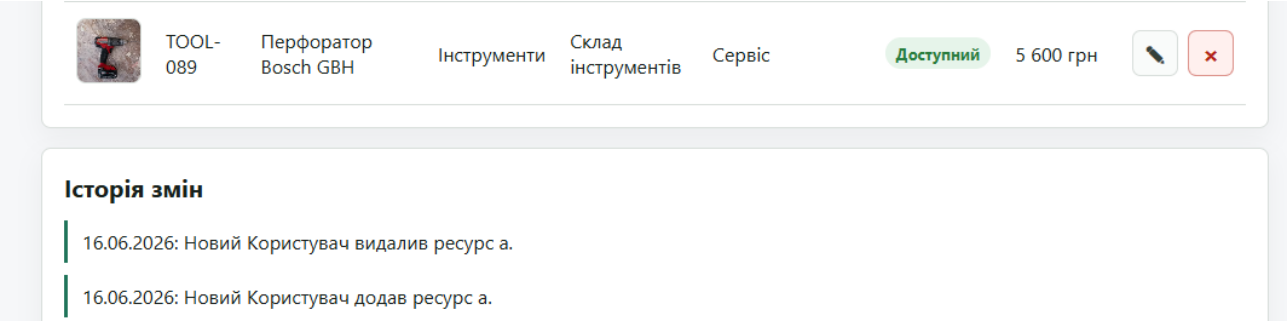


Рисунок А.3 – Видалення ресурсу з каталогу

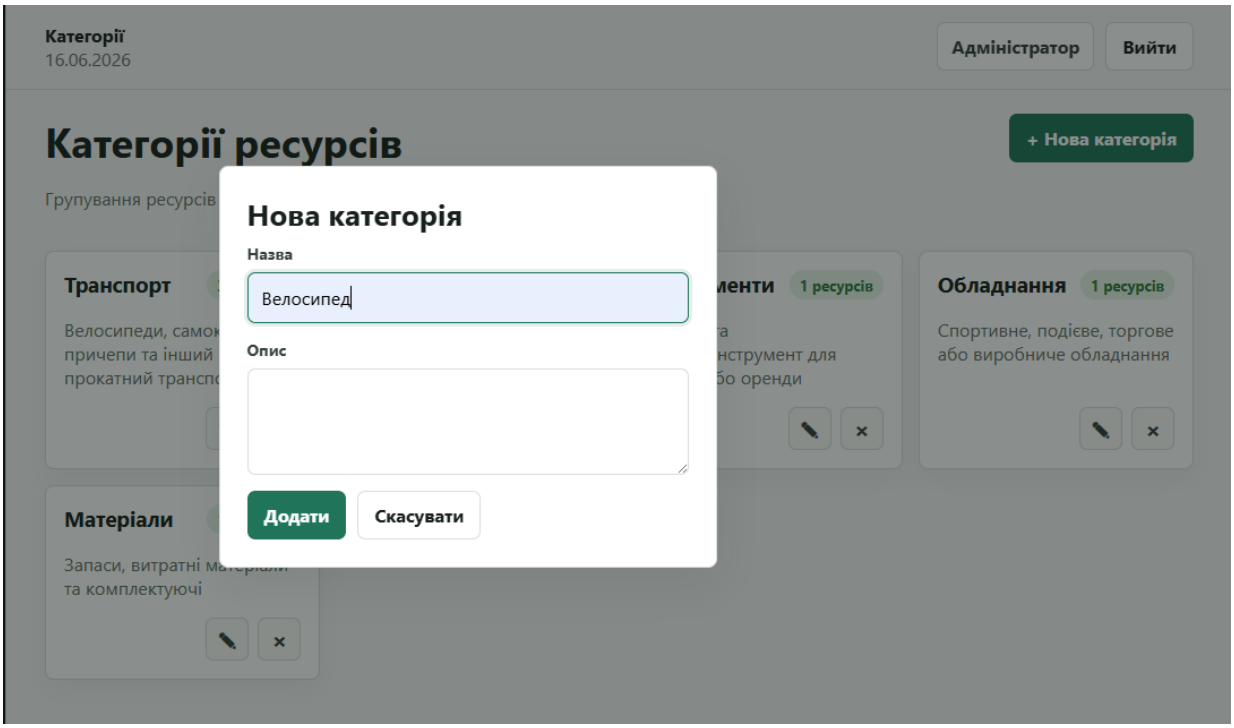


Рисунок А.4 – Додавання нової категорії ресурсів

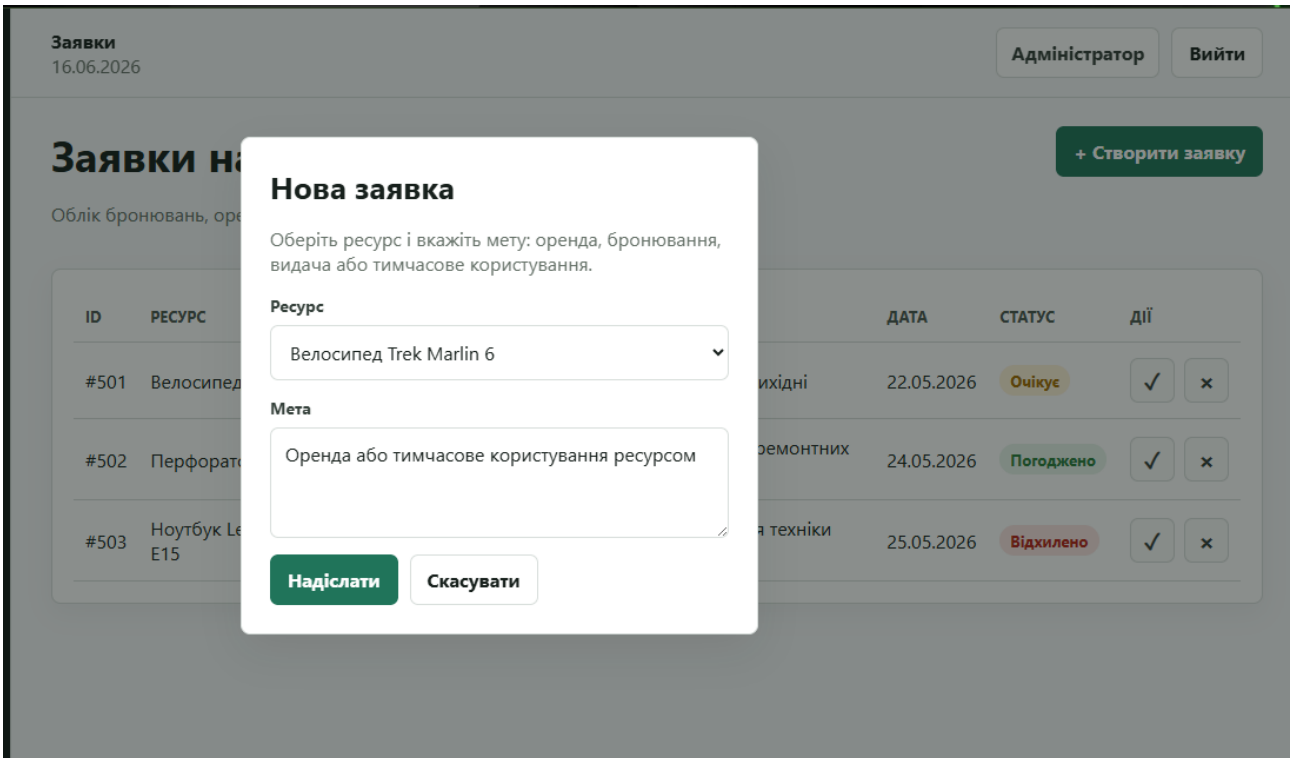


Рисунок А.5 – Створення заявки на використання ресурсу

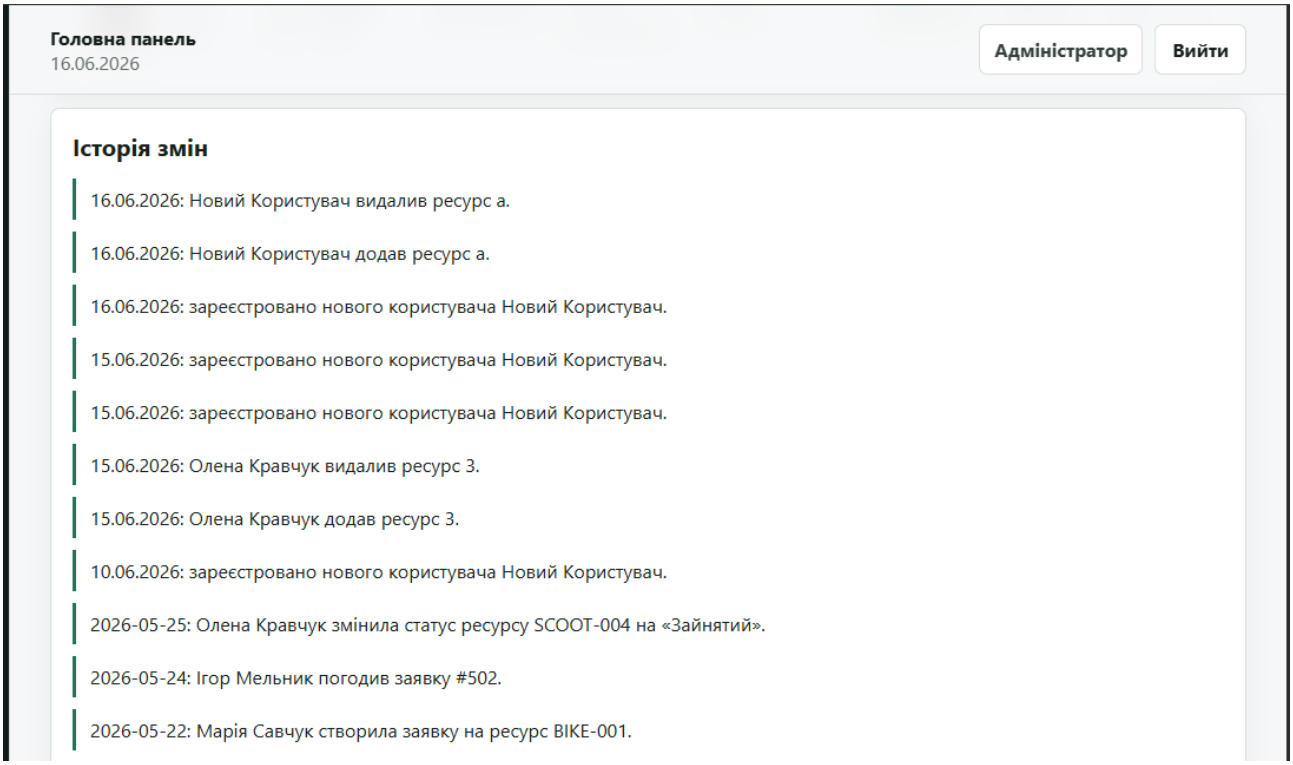
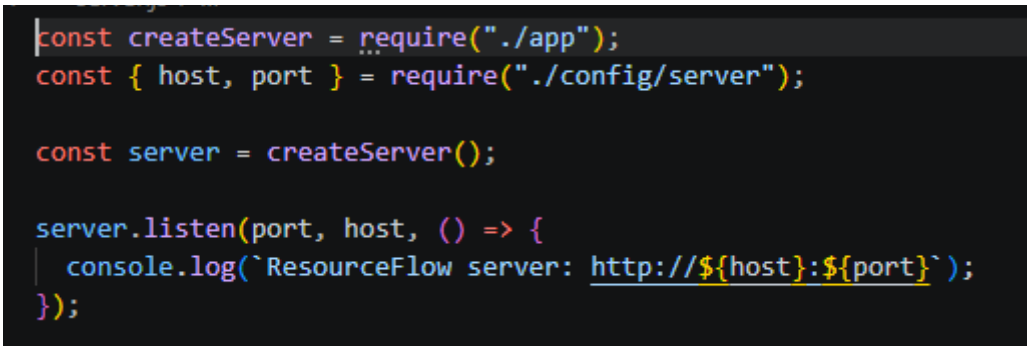


Рисунок А.6 – Перегляд звітів та історії змін у системі

ДОДАТОК Б

Фрагмент програмного коду серверної частини



```
const createServer = require("./app");
const { host, port } = require("./config/server");

const server = createServer();

server.listen(port, host, () => {
  console.log(`ResourceFlow server: http://${host}:${port}`);
});
```

Рисунок Б.1 – Фрагмент програмного коду серверної частини системи ResourceFlow

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: «Розроблення веб-орієнтованої інформаційної системи обліку ресурсів».

Обсяг пояснювальної записки: 74 аркуші

Дата закінчення бакалаврської роботи: 10 червня 2026 р.

Підпис студента _____