ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для інтеграції процесів інвентаризації та управління технічними засобами компанії»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Дмитро ПРИЛІПКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Дмитро ПРИЛІПКО

Керівник: Наталя АНДРУСЕВИЧ
ст. викладач

Рецензент:

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Приліпко Дмитру Максимовичу

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для інтеграції процесів інвентаризації та управління технічними засобами компанії »
керівник кваліфікаційної роботи старший викладач кафедри ІПЗ Наталя АНДРУСЕВИЧ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи розробки web-застосунків, опис роботи електронних бібліотек, технічна документація фреймворків Laravel та Vue.js.

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

1. Аналіз предметної галузі електронних бібліотек.
2. Вибір засобів програмної реалізації.
3. Проєктування архітектури web-застосунку.
4. Програмна реалізація та тестування web-застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма класів.
6. Діаграми активностей.
7. Діаграма впровадження.
8. Екранні форми.
9. Апробація результатів дослідження.
10. Висновки.

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд існуючих рішень	14.03-18.03.2025	
4	Огляд та аналіз засобів реалізації web-застосунку	19.03-23.03.2025	
5	Проєктування архітектури web-застосунку	24.03-13.04.2025	
6	Програмна реалізація та тестування web-застосунку	14.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Дмитро ПРИЛІПКО
(підпис)

Керівник
кваліфікаційної роботи _____ Наталя АНДРУСЕВИЧ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 52 стор., 2 табл., 32 рис., 21 джерел.

Мета роботи – покращення процесів обліку, інвентаризації та управління технічними засобами компанії шляхом створення функціонального web-застосунку.

Об'єкт дослідження – процес управління матеріальними та технічними ресурсами підприємства.

Предмет дослідження – web-застосунок для інтеграції процесів інвентаризації та управління технічними засобами компанії.

Короткий зміст роботи: В роботі проаналізовано сучасні проблеми інвентаризації та обліку технічних засобів підприємств. Проведено огляд програмного забезпечення, що використовується у цій галузі: Snipe-IT, Asset Panda, InvGate Assets, OpenMAINT. Розроблено структуру web-застосунку для автоматизації процесів обліку, інвентаризації та управління статусами технічних засобів компанії та програмно реалізовані функціональні можливості: реєстрація та авторизація користувачів, облік обладнання, інвентаризаційні списки, імпорт/експорт даних у форматах CSV/Excel. В роботі використано фреймворк Laravel для реалізації серверної логіки та Vue 3 + Vite — для розробки користувацького інтерфейсу. Для захисту даних застосовано механізми авторизації з токенами (Laravel Sanctum), а також базове шифрування паролів.

Сферою використання застосунку є автоматизація процесів обліку технічних засобів у приватних підприємствах.

КЛЮЧОВІ СЛОВА: ІНВЕНТАРИЗАЦІЯ, WEB-ЗАСТОСУНОК, LARAVEL, VUE.JS, ОБЛІК ТЕХНІКИ, API, ІМПОРТ/ЕКСПОРТ ДАНИХ

ЗМІСТ

ВСТУП

У сучасних умовах цифровізації підприємницької діяльності та зростаючої потреби в ефективному управлінні ресурсами компанії, особливого значення набуває питання автоматизації процесів інвентаризації та обліку технічних засобів. Технічні ресурси є одним із ключових елементів забезпечення стабільного функціонування будь-якої організації, а їх неефективне використання або неналежний облік може призводити до фінансових втрат, простоїв у роботі та управлінських помилок.

Проблематика, що стала основою для вибору теми дослідження, виникла у реальному корпоративному середовищі. Суттєвими викликами стали розбіжності між фактичними наявними технічними засобами та даними, що обліковувались у бухгалтерії, а також відсутність нативно зрозумілої та зручної структури візуалізації списку ресурсів для керівництва й працівників. Це ускладнювало контроль за станом техніки, відповідальними особами та потребувало модернізації процесів взаємодії з цією інформацією. Метою кваліфікаційної роботи є покращення процесів обліку, інвентаризації та управління технічними засобами компанії шляхом створення функціонального web-застосунку.

Мета роботи: покращення процесів обліку, інвентаризації та управління технічними засобами компанії шляхом створення функціонального web-застосунку.

Об’єкт дослідження: процес управління матеріальними та технічними ресурсами підприємства.

Предмет дослідження: web-застосунок для інтеграції процесів інвентаризації та управління технічними засобами компанії.

Результати цієї роботи можуть бути впроваджені в організаціях для підвищення прозорості, точності обліку та оптимізації управління технічними

ресурсами. В основі розробки покладено сучасні технології, зокрема Laravel (серверна частина), Vue.js + Vite[1] (клієнтська частина), MySQL для зберігання даних, а також інструменти взаємодії та керування станом — Axios, Vuex. Обраний технологічний стек дозволяє створити масштабовану, безпечну та зручну в користуванні систему для реального бізнес-середовища.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Приліпко Д.М., Андрусевич Н.В. ВИЗНАЧЕННЯ ВИМОГ ДО ВЕБЗАСТОСУНКА ДЛЯ ІНТЕГРАЦІЇ ПРОЦЕСІВ ІНВЕНТАРИЗАЦІЇ ТА УПРАВЛІННЯ ТЕХНІЧНИМИ ЗАСОБАМИ КОМПАНІЇ З ВИКОРИСТАННЯМ ФРЕЙМВОРКІВ VUE.JS ТА LARAVEL. // Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, ДУІКТ, Київ, Україна, с. 74-75

2. Приліпко Д.М., Андрусевич Н.В. БЕЗПЕКА ВЕБЗАСТОСУНКУ ДЛЯ ІНТЕГРАЦІЇ ПРОЦЕСІВ ІНВЕНТАРИЗАЦІЇ ТА УПРАВЛІННЯ ТЕХНІЧНИМИ ЗАСОБАМИ КОМПАНІЇ З ВИКОРИСТАННЯМ ФРЕЙМВОРКІВ VUE.JS ТА LARAVEL. // Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025». 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с. 317-319.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Web-застосунки для автоматизації інвентаризаційних процесів

Інформаційні системи, що забезпечують автоматизацію інвентаризації та управління технічними засобами, є важливою складовою сучасної цифрової інфраструктури підприємств, установ та організацій. Такі web-застосунки дозволяють систематизувати та централізувати облік матеріальних ресурсів, спрощують процес інвентаризації, забезпечують точність даних та зменшують витрати часу і людських ресурсів на обробку інформації.

Сучасні платформи для інвентаризації вирішують низку типових проблем, серед яких: неузгодженість інформації між обліковими системами та фактичною наявністю активів, відсутність оперативного доступу до актуальних даних про місцезнаходження, стан і відповідальних осіб за обладнання, складність підготовки звітів та аналізу використання ресурсів. Вони також дозволяють уникнути дублювання записів, втрати об’єктів обліку, а також надають гнучкі засоби фільтрації, сортування, групування та експорту інформації.

Інвентаризаційні web-застосунки здебільшого працюють за принципом клієнт-серверної архітектури. Користувачі через браузер взаємодіють із фронтендом системи, який надає доступ до функцій додавання, редагування, пошуку, фільтрації та перегляду активів. Серверна частина забезпечує обробку запитів, автентифікацію, логіку обробки даних, валідацію та зберігання інформації у базі даних. Такі рішення легко масштабуються, інтегруються з іншими системами, наприклад бухгалтерськими або ERP-платформами, та адаптуються під специфіку кожної організації.

Загалом застосування спеціалізованих web-застосунків для обліку технічних засобів дозволяє підвищити ефективність управління ресурсами, знизити ризики втрат, покращити контроль та прозорість усіх операцій з активами.

1.2 Цільова аудиторія

Цільовою аудиторією застосунків для автоматизації інвентаризації та управління технічними засобами є підприємства малого та середнього бізнесу, організації, які мають в обігу велику кількість технічних засобів, а також державні установи, де необхідний контроль за обладнанням, закупівлями та інвентаризацією. Наприклад:

- керівники IT-відділів зможуть оперативно переглядати стан техніки, переглядати розподіл по відповідальним особам та готувати звіти;
- системні адміністратори відповідальні за оновлення інформації щодо технічних засобів, їхнього обслуговування та списання;
- фінансово-бухгалтерські служби можуть здійснювати перехресну перевірку фактичного стану техніки з бухгалтерськими записами;
- офіс-менеджери та матеріально-відповідальні особи мають змогу здійснювати швидкий пошук техніки за кабінетами чи відділами;
- інвентаризаційні комісії використовують застосунок для формування та ведення актів перевірки.

Таким чином, застосунки для автоматизації інвентаризації та управління технічними засобами орієнтовані на широке коло користувачів і дозволяють забезпечити прозорий і ефективний облік усіх технічних засобів компанії.

1.3 Аналіз аналогів

У процесі проєктування програмного забезпечення було проведено аналіз сучасних рішень, які мають подібний функціонал. До основних аналогів можна віднести такі системи, як Snipe-IT, Asset Panda, InvGate Assets, OpenMAINT. Розглянемо їх детальніше.

1.3.1 Snipe-IT

Snipe-IT[2] — це безкоштовна система з відкритим вихідним кодом, яка широко використовується для обліку активів у невеликих і середніх компаніях. Вона дозволяє здійснювати облік обладнання, відстеження змін у статусі, призначення відповідальних осіб, контроль ліцензійного програмного забезпечення, ведення історії переміщень техніки тощо. Система має web-інтерфейс, REST API, підтримку ролей і дозволів. Приклад інтерфейсу додатку наведено на рисунку 1.1.

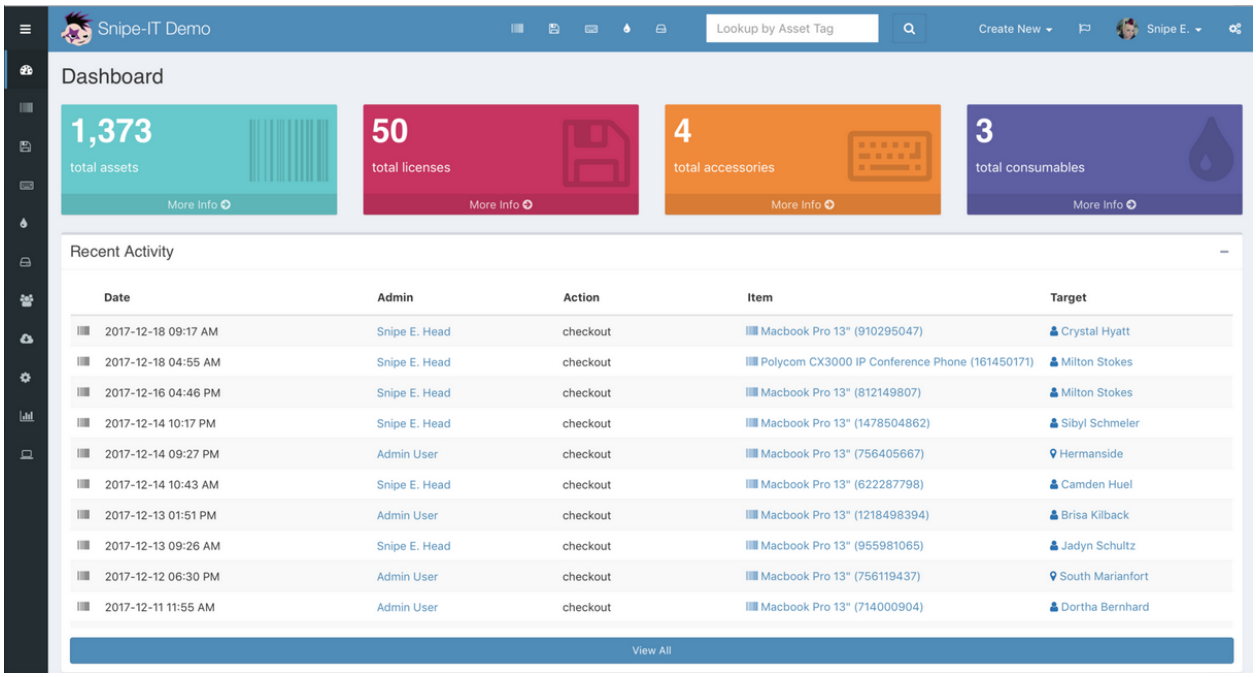


Рис. 1.1 Інтерфейс додатку Snipe-IT

Переваги:

- повністю безкоштовна, open-source ліцензія;
- відкритий код дозволяє розробникам вносити власні зміни;
- має функції відстеження історії активів та логування змін;
- підтримка API для інтеграції з іншими сервісами.

Недоліки:

– обмежені можливості кастомізації інтерфейсу без значного втручання в код;
– відсутність багатомовної підтримки (інтерфейс англійською);
– вимоги до серверного розгортання та обслуговування.

1.3.2 Asset Panda

Asset Panda[3] — це комерційна SaaS-платформа для обліку активів, орієнтована переважно на великий бізнес. Вона пропонує розширені можливості: мобільний доступ, штрихкодування, нагадування про техобслуговування, інтеграції з бухгалтерськими системами. Приклад інтерфейсу додатку наведено на рисунку 1.2.

Рис. 1.2 Інтерфейс додатку Asset Panda

Переваги:

– гнучке налаштування атрибутів та звітів;
– хмарне середовище, доступ з будь-якого пристрою;
– розширена технічна підтримка та документація;

– підтримка командної роботи, нотаток, коментарів, вкладених файлів.

Недоліки:

– висока вартість ліцензії, що ускладнює впровадження у невеликих компаніях;
– частина функціоналу доступна лише в розширених тарифах;
– переважно англомовний інтерфейс і документація.

1.3.3 InvGate Assets

InvGate Assets[4] — потужна система з фокусом на аналітику та автоматизацію процесів, яка орієнтована на середні та великі компанії. Вона підтримує моніторинг обладнання, управління ліцензіями, автоматичне виявлення пристроїв у мережі. Приклад інтерфейсу додатку наведено на рисунку 1.3.

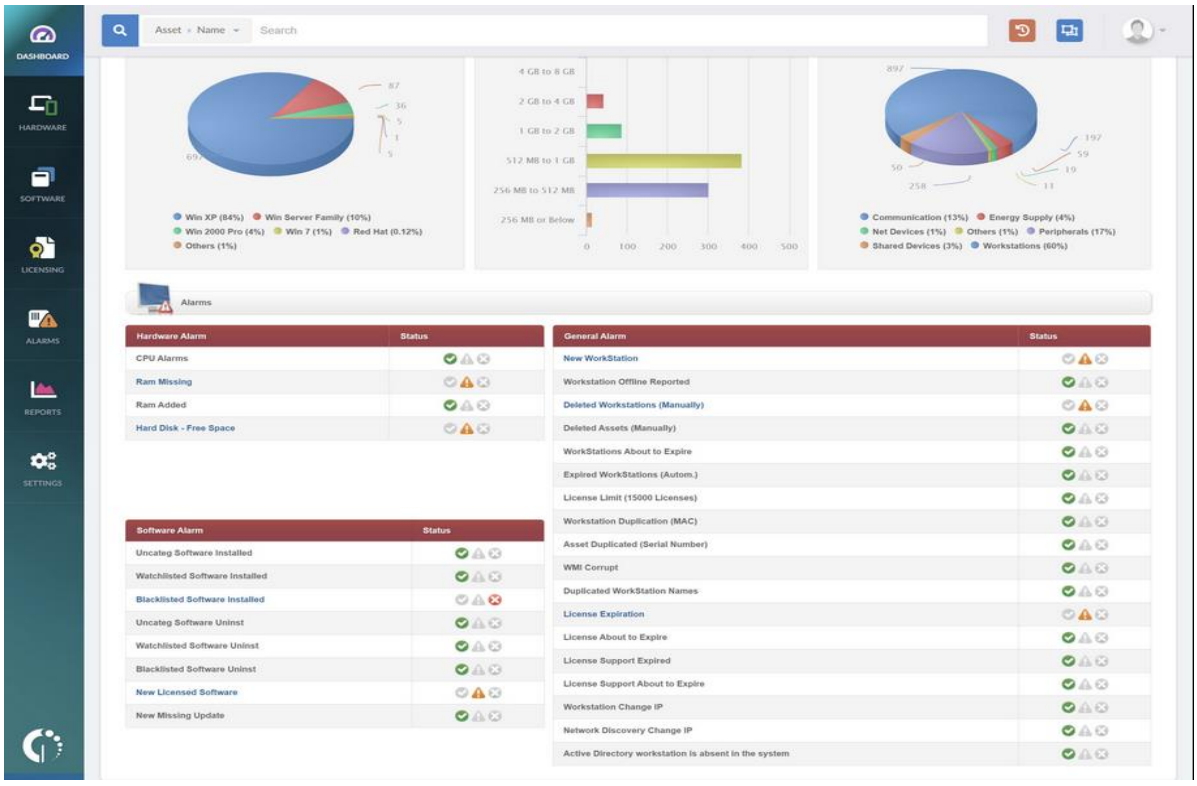


Рис. 1.3 Інтерфейс додатку InvGate Assets

Переваги:

– розвинуті аналітичні інструменти (дашборди, графіки);
– інтеграція з Active Directory, з системами helpdesk;
– підтримка різних ОС і типів пристроїв;
– високий рівень безпеки та аудиту дій.

Недоліки:

– складне впровадження та налаштування, особливо для середніх підприємств;
– висока ціна використання, особливо при масштабуванні;
– обмежена локалізація.

1.3.4 OpenMAINT

OpenMAINT[5] — open-source платформа, створена для управління нерухомістю, інфраструктурою, інвентарем і обладнанням. Незважаючи на широкі можливості, її головний фокус — облік будівель, комунікацій та інженерних мереж, а не дрібної техніки. Приклад інтерфейсу додатку наведено на рисунку 1.4.

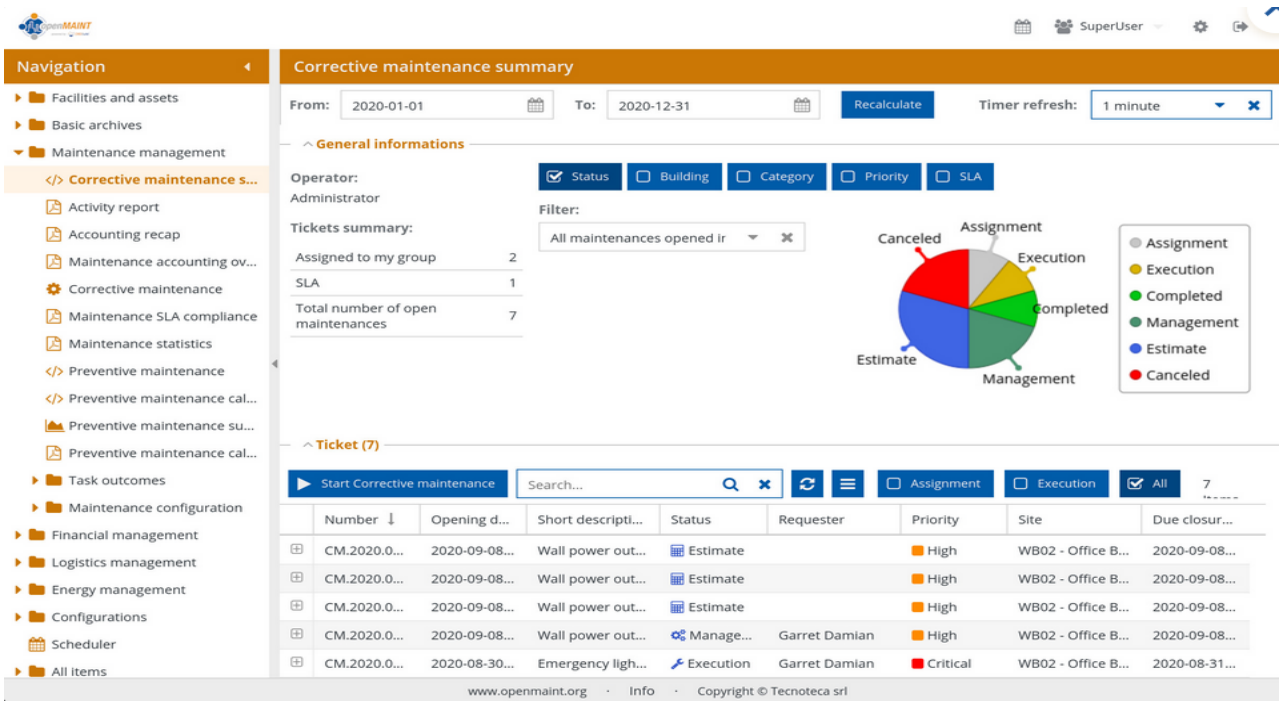


Рис. 1.4 Інтерфейс додатку OpenMAINT

Переваги:

– модульна структура, можливість розширення;
– ведення повного життєвого циклу об'єктів;
– підтримка документації, планів, інтеграцій.

Недоліки:

– складність у використанні для обліку лише IT-активів;
– надлишкова для задач інвентаризації обладнання;
– потребує навичок роботи з PostgreSQL, Java і OpenLayers.

Зведені результати аналізу характеристик розглянутих додатків наведено у таблиці 1.1.

Таблиця 1.1

Зведені результати аналізу характеристик додатків для контролю персональної економічної активності

Показник/ Застосунок	Snipe-IT	Asset Panda	InvGate Assets	OpenMAINT	EVR(Власна розробка)
Ліцензія	Open Source	Commercial	Commercial	Open Source	Commercial
Хмарне середовище	-	+	+	-	+
API / інтеграції	+	+	+	+	+
Кастомізація	-	-	-	-	+
Придатність до малих підприємств	+	-	-	-	+
Локалізація / підтримка мов	Англійська	Англійська	Англійська	Англійська	Англійсько-Українська
Простота впровадження	Середня	Висока	Низька	Низька	Висока

2 ЗАСОБИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

2.1 Середовище розробки

2.1.1 WebStorm

WebStorm[6] — це одна з найпотужніших інтегрованих серед розробки (IDE) для JavaScript, створена компанією JetBrains. Вона пропонує широкий набір інструментів для роботи з вебтехнологіями: підтримку фреймворків Vue, React, Angular, перевірку синтаксису в реальному часі, рефакторинг, автодоповнення, відлагодження, інтеграцію з Git та CI/CD-платформами. WebStorm спрощує роботу з сучасними SPA, підтримує модульну структуру проєкту та дозволяє зменшити кількість помилок на ранніх етапах розробки. Завдяки глибокій інтеграції з екосистемою Node.js і високому рівню автоматизації типових завдань розробки, ця IDE значно підвищує продуктивність і комфорт роботи розробника.

У межах даної роботи WebStorm використовувалася як основне середовище для розробки клієнтської частини. Особливо корисною виявилася функція автоматичного оновлення інтерфейсу (hot-reload), яка дозволяла вносити зміни в компоненти Vue.js та одразу бачити результат у браузері без необхідності перезавантаження сторінки. Також активно використовувалася інтеграція з Git для керування версіями та спрощення роботи з гілками: розробник мав можливість здійснювати коміти, злиття, перегляд історії змін безпосередньо з інтерфейсу IDE, що значно прискорювало робочий процес. WebStorm також автоматично підсвічував помилки синтаксису, допомагав у рефакторингу коду та підтримував автодоповнення для API браузера, бібліотек і модулів, що було надзвичайно корисним при роботі з великими проєктами на Vue.js. Приклад інтерфейсу середовища розробки WebStorm наведено на рисунку 2.1.

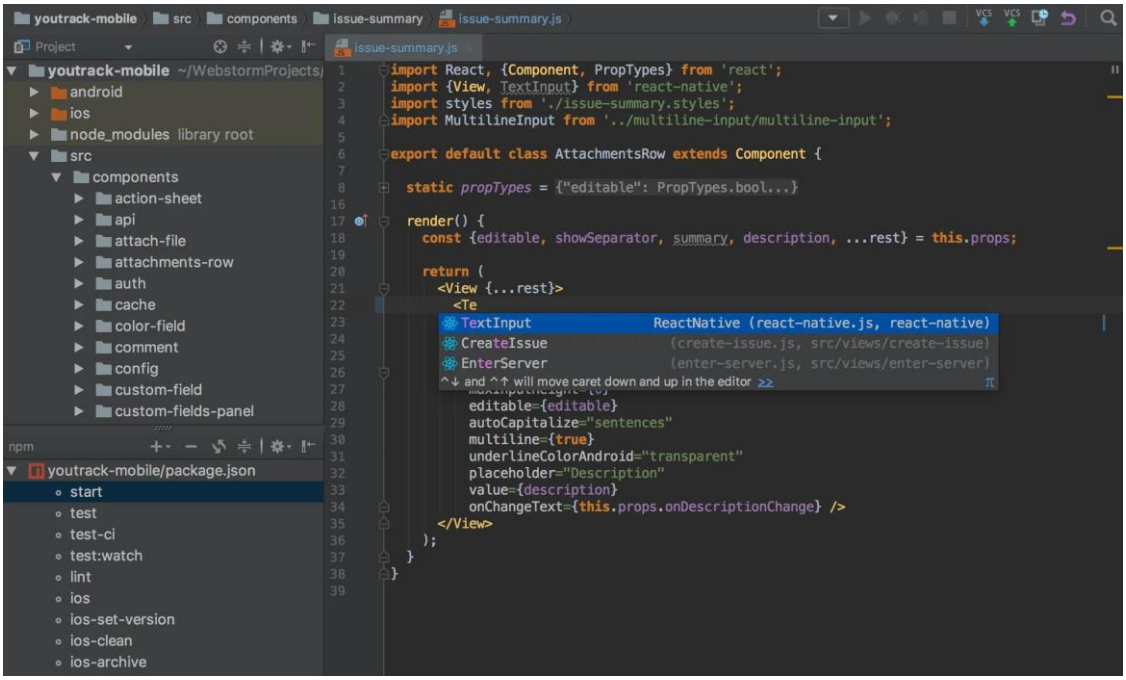


Рис. 2.1 Інтерфейс середовища розробки WebStorm

2.1.2 GitLab

GitLab[7] — це повноцінна DevOps-платформа з відкритим кодом, яка поєднує засоби контролю версій, управління проєктами, CI/CD, документацією, безпекою та автоматизацією в єдиному середовищі. У роботі GitLab використовується для ведення репозиторію коду, організації розробки в гілках, коду-рев'ю через Merge Requests, автоматизованого тестування та збирання застосунку. Це забезпечує прозорість процесу розробки, зменшення людського фактора та високу надійність коду. Крім того, платформа дозволяє налаштовувати середовище для кожного розробника, вести облік змін, відстежувати баги та координувати командну роботу, що робить її незамінним інструментом для сучасної розробки.

Однією з ключових переваг GitLab є підтримка CI/CD-пайплайнів, які дозволяють автоматизувати процеси збирання, тестування та розгортання коду. У межах роботи було створено власний .gitlab-ci.yml файл, у якому описані етапи

перевірки синтаксису, збирання фронтенду, виконання Laravel-команд (migrate, config:cache, route:clear) і деплой на продакшн-сервер. GitLab CI виконує всі ці операції автоматично при кожному оновленні коду в основній гілці.

Також GitLab активно використовувався для командної роботи над проєктом. Функція Merge Requests дозволяла здійснювати рев'ю коду перед злиттям, обговорювати реалізацію функціоналу в коментарях, перевіряти результати автоматичних перевірок. Такий підхід суттєво підвищував якість коду, забезпечував дотримання стилю програмування та виявляв помилки на ранніх етапах.

Для історії змін GitLab автоматично фіксує всі коміти, візуалізує різницю між версіями, дозволяє швидко повернутися до попередньої ревізії, що є надзвичайно корисним у випадках виникнення багів або необхідності відновлення стабільної версії проєкту. Крім того, GitLab інтегрується з системами моніторингу та нотифікацій, що дозволяє оперативно отримувати сповіщення про збої або зміни в пайплайні.

У даній роботі GitLab не лише виступає сховищем коду, а є центром усіх етапів життєвого циклу розробки — від створення задач до автоматизованого тестування і доставлення змін у продакшн-середовище. Приклад інтерфейсу платформи GitLab наведено на рисунку 2.2.

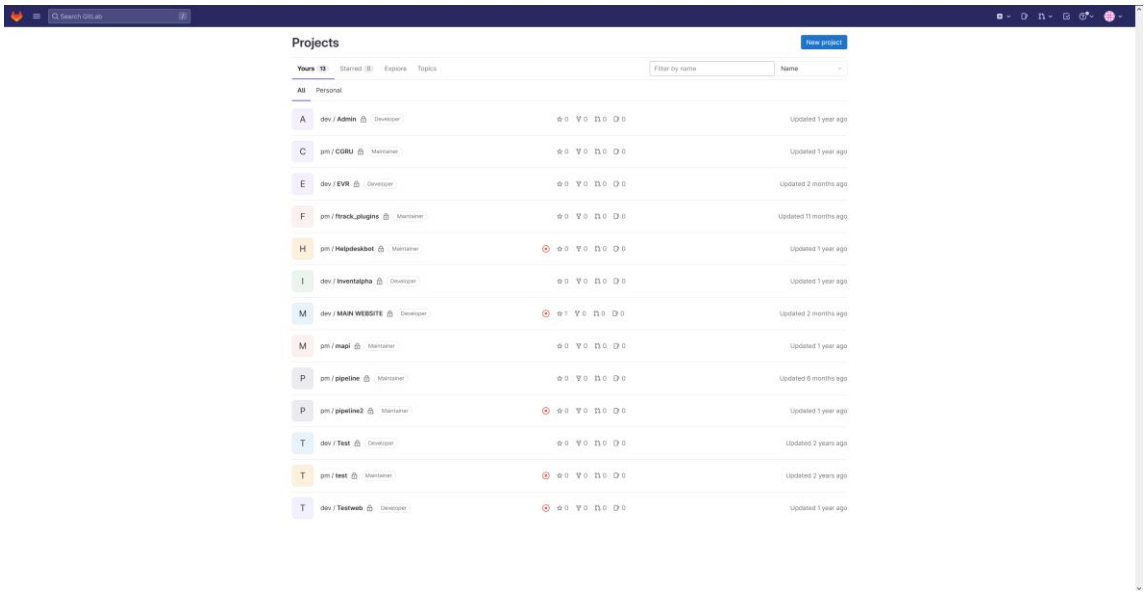


Рис. 2.2 Інтерфейс платформи GitLab

2.2 Мови програмування

2.2.1 PHP

PHP[8] — це мова програмування, спеціально створена для веброзробки, яка активно підтримується спільнотою протягом багатьох років. Вона була розроблена у 1994 році Рамусом Лердорфом і з того часу зазнала значного розвитку. Станом на 2024 рік, згідно з даними W3Techs, PHP використовується більш ніж на 75% усіх вебсайтів, серверна частина яких базується на мовах програмування. Серед найвідоміших платформ, що використовують PHP, — Facebook (на ранньому етапі), Wikipedia, WordPress, Drupal, Joomla.

Однією з головних переваг PHP є її простота в освоєнні, велика кількість готових рішень і широка спільнота, що постійно підтримує та розвиває мову. PHP дозволяє створювати як прості сайти, так і складні web-застосунки з інтеграцією баз даних, REST API, механізмами авторизації, а також компонентами безпеки. Завдяки великій кількості доступних фреймворків — зокрема Laravel, Symfony, Yii — PHP зберігає свою актуальність і високу конкурентоспроможність у сучасному вебсередовищі.

У даній роботі PHP використовується як основна серверна мова, що забезпечує надійну взаємодію між клієнтом і базою даних, реалізацію API та зв’язок з системами авторизації на основі Active Directory. У поєднанні з фреймворком Laravel вона дозволяє ефективно організувати бізнес-логіку, маршрутизацію та захист запитів, забезпечуючи при цьому стабільну та швидку роботу застосунку.

2.2.2 JavaScript

JavaScript[9] — мова програмування, яка є основою для створення інтерактивних елементів на вебсторінках. Вона з’явилася у 1995 році та швидко стала стандартом де-факто для фронтенд-розробки. Завдяки універсальності та широкій підтримці у браузерах JavaScript отримав стрімкий розвиток і вийшов далеко за межі клієнтської логіки. Згодом, із появою середовища Node.js, мова

стала активно використовуватися і для серверної розробки, що зробило її повноцінним інструментом для створення повнофункціональних web-застосунків.

На 2024 рік JavaScript залишається найбільш поширеною мовою програмування в інтернеті: згідно зі статистикою Stack Overflow, понад 65% розробників регулярно її використовують у професійній діяльності. Це пояснюється широким набором інструментів, фреймворків і бібліотек, серед яких Vue.js, React, Angular, Express.js та інші. Мова підтримує об’єктно-орієнтовану, функціональну й реактивну парадигми, має гнучкий синтаксис і велике співтовариство, яке постійно її вдосконалює.

У межах даної роботи JavaScript використовується для реалізації клієнтської частини за допомогою фреймворку Vue.js. Він дозволяє створювати динамічний інтерфейс користувача, де сторінки завантажуються один раз, а подальша взаємодія відбувається без перезавантаження. Це забезпечує високу швидкодію, зручність для користувача та мінімальне навантаження на сервер. JavaScript дозволив реалізувати асинхронну взаємодію з API через Axios, керування станом додатка через Vuex, маршрутизацію через Vue Router, а також обробку подій, валідацію форм та динамічне оновлення даних в інтерфейсі.

Гнучкість, багата екосистема, постійна підтримка браузерами та активна спільнота роблять JavaScript незамінною мовою в сучасній веброзробці. Його застосування у цій роботі стало ключовим чинником у створенні зручного, адаптивного та продуктивного користувацького інтерфейсу. для створення інтерактивних елементів на вебсторінках. Вона працює безпосередньо в браузері, що дозволяє реалізовувати динамічну взаємодію з користувачем у реальному часі. Сьогодні JavaScript використовується як на фронтенді, так і на бекенді, завдяки розвитку платформ типу Node.js. У цій роботі JavaScript використовується у зв’язці з Vue.js для побудови реактивного інтерфейсу. Серед переваг мови — легкість у вивченні, активна екосистема, сумісність з усіма браузерами та велика кількість фреймворків і бібліотек, що значно прискорюють розробку складних SPA-застосунків.

2.3 Серверна частина

2.3.1 Laravel

Laravel[10] — це популярний і сучасний фреймворк для PHP, який був створений у 2011 році Тейлором Отвеллом і з того часу активно розвивається та підтримується великою міжнародною спільнотою. Його основною метою є спрощення процесу розробки web-застосунків, підвищення читаємості коду, впровадження найкращих практик архітектури ПЗ та забезпечення високого рівня безпеки. Laravel побудований на основі архітектурного шаблону MVC (Model-View-Controller), який дозволяє чітко розділяти відповідальність між рівнями обробки даних, логіки та представлення.

До основних складових Laravel належать: маршрутизація з підтримкою REST-підходу, потужна ORM-система Eloquent для роботи з базами даних, система шаблонів Blade для побудови інтерфейсів, механізм middleware для проміжної обробки запитів, інтегровані механізми автентифікації й авторизації, вбудована підтримка подій, черг, обробників та планувальника завдань (Scheduler). Завдяки інтуїтивному API, Laravel дозволяє легко реалізовувати розширені можливості навіть початківцям, при цьому зберігаючи гнучкість для складних корпоративних проєктів.

Фреймворк активно підтримує принципи безпечного програмування: зокрема, має захист від SQL-ін’єкцій, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) та забезпечує надійне зберігання паролів через сучасні алгоритми хешування. Laravel також має розгалужену екосистему офіційних пакетів, таких як Laravel Horizon (для моніторингу черг), Laravel Nova (адмін-панель), Laravel Passport і Sanctum (автентифікація), Laravel Scout (пошук) тощо.

Важливою рисою є підтримка інфраструктурного рівня: Laravel надає зручні засоби для роботи з міграціями баз даних, сидуванням тестових даних, написанням юніт- та інтеграційних тестів. Усе це робить його не лише фреймворком для розробки, а й середовищем повного циклу — від початку створення проєкту до його розгортання та підтримки., який базується на архітектурному шаблоні MVC (Model-View-Controller) і надає розробнику інструменти для створення безпечних, масштабованих і структурованих web-застосунків. Laravel включає в себе компоненти для маршрутизації,

шаблонізації (Blade), роботи з базами даних через ORM Eloquent, підтримку middleware, валідацію даних, черги завдань, події та багато іншого. Перевагою є також наявність зручного механізму для написання REST API та високий рівень безпеки завдяки вбудованому захисту від CSRF, XSS і SQL-ін'єкцій. Laravel має розвинену екосистему (Horizon, Nova, Forge) та активну спільноту, що значно полегшує розробку і підтримку.

2.3.2 Laravel Sanctum

Laravel Sanctum — це легковагий пакет для Laravel, який забезпечує автентифікацію користувачів через API токени або cookies[12]. Його основною особливістю є простота інтеграції та підтримка SPA-застосунків. Sanctum дозволяє захищати маршрути, обмежувати доступ до ресурсів, прив'язувати токени до пристроїв і встановлювати їхні терміни дії. Він забезпечує базовий рівень безпеки та сумісність з іншими механізмами автентифікації Laravel без потреби у складній конфігурації.

2.3.3 CSRF-токен

CSRF (Cross-Site Request Forgery) — один із найпоширеніших типів атак на web-застосунки, коли зловмисник намагається змусити браузер користувача надіслати небажаний запит. Laravel автоматично реалізує захист від подібних атак шляхом генерації унікального токена для кожної сесії користувача. Цей токен додається до кожної форми або AJAX-запиту, а сервер перевіряє його на автентичність. Завдяки цьому механізму підвищується безпека даних користувачів і захищається система від автоматизованих атак.

2.3.4 Авторизація через Active Directory

Active Directory[11] (AD) — це служба каталогів, розроблена компанією Microsoft, яка дозволяє централізовано керувати обліковими записами користувачів, комп'ютерів, групами, політиками безпеки та правами доступу в масштабі організації. Вона є невід'ємною частиною багатьох корпоративних інфраструктур і використовується для автентифікації та авторизації працівників

при доступі до внутрішніх сервісів, включно з web-застосунками.

Інтеграція авторизації через Active Directory у web-застосунок дозволяє реалізувати єдиний вхід (SSO), при якому користувач вводить свої корпоративні облікові дані один раз і отримує доступ до всіх сервісів, на які має відповідні дозволи. Це значно підвищує рівень зручності для користувачів і зменшує кількість запитів до адміністратора на створення локальних акаунтів.

У межах даної роботи реалізовано механізм перевірки облікових даних користувача шляхом запиту до Active Directory. Якщо користувач існує в домені та належить до певної групи (наприклад, IT-відділу або адміністративного персоналу), йому надається доступ до функціоналу системи. Додатково система може здійснювати перевірку доменного імені користувача, після чого генерується CSRF-токен, що дозволяє підтримувати безпечну сесію.

Використання AD дозволяє підвищити інформаційну безпеку — завдяки централізованому управлінню паролями, політиками зміни паролів, журналюванням входів і виходів. Також це дає змогу гнучко керувати правами доступу без потреби змінювати бізнес-логіку застосунку: достатньо змінити роль користувача або групу в AD. Система є масштабованою і може інтегруватися з іншими службами аутентифікації — наприклад, Kerberos, LDAP, або сучасними протоколами OpenID та SAML.

2.4 Клієнтська частина

2.4.1 Vue.js

Vue.js[13] — це прогресивний JavaScript-фреймворк для створення інтерфейсів користувача, який був розроблений Еваном Ю у 2014 році. З моменту свого запуску Vue швидко набув популярності серед розробників завдяки простоті, гнучкості та потужності. Фреймворк поєднує в собі найкращі риси своїх конкурентів, таких як Angular і React, але водночас зберігає легкість входження для початківців і гнучкість для великих корпоративних проєктів.

Vue побудований за компонентною архітектурою, яка дозволяє розбивати інтерфейс на незалежні, ізольовані блоки (компоненти). Це полегшує повторне використання коду, тестування, супровід і масштабування застосунку. Усі компоненти можуть містити власну розмітку (HTML), логіку (JavaScript) та стилі (CSS), що забезпечує зручність при розробці навіть великих інтерфейсів.

Однією з ключових переваг Vue є його реактивність: система автоматично відстежує зміни в даних і оновлює інтерфейс, не потребуючи додаткових викликів рендерингу. Завдяки цьому користувач бачить оновлення в режимі реального часу, що особливо корисно у випадках інтерактивної роботи з таблицями, формами чи дашбордами.

Фреймворк підтримує двостороннє зв’язування даних (two-way binding), що дає змогу синхронізувати модель і представлення без проміжного коду.

Також Vue має декларативний синтаксис шаблонів, що робить логіку рендерингу зрозумілою та прозорою. Інтеграція з сучасними інструментами, такими як Vite, Babel, ESLint, робить процес розробки швидким та контрольованим.

Vue.js має активну спільноту, багату документацію, офіційні додаткові модулі (Vue Router для маршрутизації, Vuex для керування станом, Pinia як сучасна альтернатива Vuex, Vue CLI для старту проєктів) та широку підтримку екосистеми компонентів. Його легко інтегрувати як у нові, так і в існуючі проєкти, включно з поетапною міграцією старих інтерфейсів.

У межах даної роботи вибір Vue.js був обумовлений потребою створення швидкого, реактивного, легкого в супроводі інтерфейсу, здатного ефективно обробляти велику кількість даних і забезпечувати сучасний користувацький досвід. для створення інтерфейсів користувача, який був розроблений Еваном Ю у 2014 році. Vue поєднує в собі гнучкість, простоту й ефективність. Його архітектура заснована на компонентному підході, що дозволяє легко масштабувати застосунок і спрощує супровід коду.

Основною перевагою Vue є реактивна система, яка дозволяє автоматично оновлювати інтерфейс при зміні даних. Завдяки підтримці шаблонів,

декларативному зв’язуванню даних і двосторонньому зв’язку, Vue дозволяє швидко створювати динамічні SPA-застосунки. У поєднанні з Vite фреймворк забезпечує миттєву збірку і гаряче оновлення модулів, що значно покращує досвід розробки.

2.4.2 Vuex

Vuex[14] — це офіційна бібліотека для керування станом у застосунках на Vue.js. Вона побудована на основі концепцій Flux і дозволяє централізовано зберігати всі змінні, які можуть бути потрібні багатьом компонентам.

Vuex надає структуру, яка полегшує відлагодження та тестування, і забезпечує передбачуваність у зміні стану, що є надзвичайно важливим у великих SPA-додатках. Його інтеграція з Vue робить процес зв’язування даних логічним і керованим, а завдяки модульності можна легко масштабувати структуру сховища при зростанні проєкту.

2.4.3 Axios

Axios[15] — це популярна JavaScript-бібліотека для виконання HTTP-запитів. Вона побудована на базі промісів і забезпечує простий та гнучкий API для взаємодії з RESTful API. Axios підтримує роботу з токенами, автоматичне перехоплення запитів і відповідей, обробку помилок, таймаути, а також дозволяє налаштовувати заголовки для кожного запиту.

У роботі Axios використовується для зв’язку між фронтендом та серверною частиною, що забезпечує ефективну взаємодію в рамках SPA. Перевага Axios також полягає у його сумісності з бібліотеками для SSR (серверного рендерингу) та простоті інтеграції з компонентами Vue.js.

2.5 База даних MySQL

MySQL[16] — це одна з найпоширеніших реляційних систем управління базами даних, розроблена ще у 1995 році. Вона є проєктом з відкритим кодом, який підтримується компанією Oracle. MySQL дозволяє зберігати великі обсяги структурованих даних і підтримує мову запитів SQL.

Серед її переваг — висока продуктивність, надійність, підтримка транзакцій, збережених процедур, реплікації, індексування та зовнішніх ключів. У поєднанні з ORM Eloquent від Laravel MySQL дозволяє створювати складні зв’язки між таблицями, забезпечувати цілісність даних і формувати запити до БД у зручному об’єктно-орієнтованому стилі.

У межах роботи база даних використовується для зберігання інформації про користувачів, обладнання, інвентаризації, кабінети, статуси, а також для ведення журналу змін. Таким чином, використаний стек технологій дозволяє реалізувати ефективний, масштабований і безпечний web-застосунок для управління технічними засобами підприємства.

3 ПРОЄКТУВАННЯ

3.1 Вимоги до програмного забезпечення

Функціональні вимоги:

- реєстрація та авторизація користувачів: можливість реєструватися та входити в систему, призначення ролей;
- облік технічних засобів: користувач може додавати нове обладнання, редагувати інформацію, прив’язувати обладнання до відповідального співробітника або кабінету;
- інвентаризація: можливість формувати списки для проведення інвентаризації, позначення фактичної наявності, стану, місцезнаходження;
- управління статусами: відображення стану обладнання: використовується, на складі, в ремонті, списано;
- експорт та імпорт даних: підтримка форматів Excel/CSV для обміну інформацією.

Нефункціональні вимоги:

- захист персональних даних користувачів, шифрування паролів;
- можливість розширення функціоналу;
- оптимізація запитів до бази даних для великого обсягу інформації;
- відгук системи не більше ніж 2-3 секунд.

Діаграму варіантів використання наведено на рисунку 3.1.

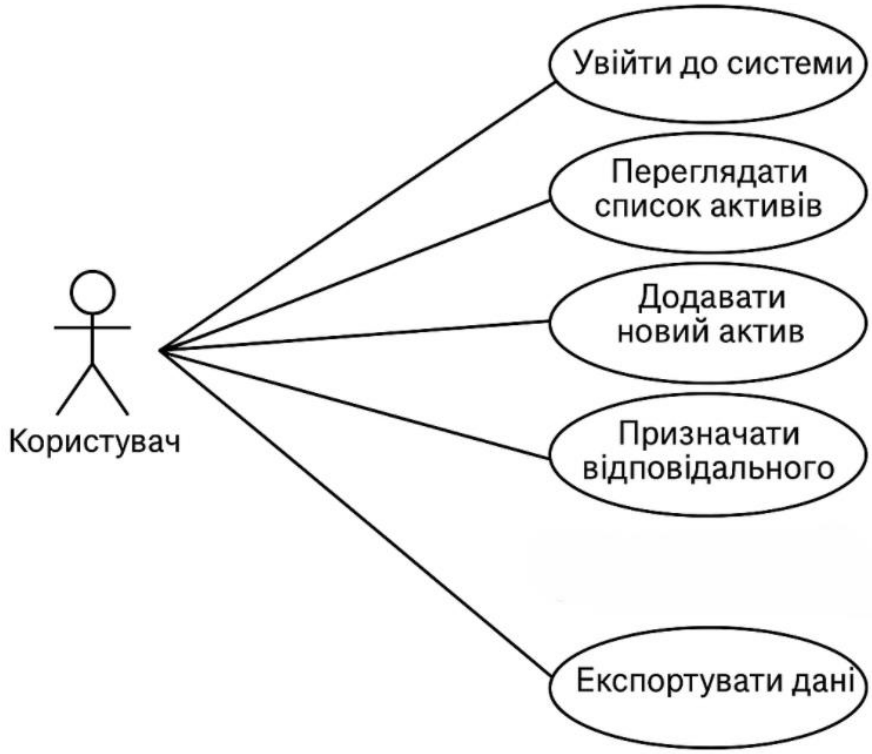


Рис. 3.1 Діаграма варіантів використання

3.2 Проєктування архітектури застосунку

Проєктування архітектури застосунку є одним із найвідповідальніших етапів створення інформаційної системи. У межах цієї системи було прийнято рішення реалізувати архітектуру за класичною клієнт-серверною моделлю. Такий підхід дозволяє розділити відповідальність між клієнтською та серверною частинами, що сприяє підвищенню масштабованості, продуктивності та зручності супроводу програмного забезпечення.

Клієнтська частина реалізована як SPA (Single Page Application) за допомогою фреймворку Vue.js. Вона відповідає за відображення інформації, взаємодію з користувачем, виконання асинхронних запитів до серверу, обробку відповідей API та підтримку стану інтерфейсу. Інтерфейс працює динамічно без перезавантаження сторінок, що покращує користувацький досвід і знижує навантаження на сервер.

Серверна частина написана з використанням PHP-фреймворку Laravel, який забезпечує логіку бізнес-процесів, перевірку даних, автентифікацію користувачів, обробку запитів та взаємодію з базою даних. Laravel також відповідає за реалізацію API, через які клієнтська частина взаємодіє з сервером.

Архітектура застосунку є RESTful, тобто використовує HTTP-методи (GET, POST, PUT, DELETE) для взаємодії з ресурсами.

Інфраструктурна частина системи базується на платформі віртуалізації Proxmox, що дозволяє створювати ізольовані середовища для серверної, клієнтської частин і бази даних.

Це забезпечує стабільність роботи системи, її гнучке масштабування, а також спрощує адміністрування. Вебсервер NGINX виконує роль реверс-проксі, перенаправляючи запити до відповідних служб та забезпечуючи безпечне з’єднання між користувачем і сервером.

Процеси безперервної інтеграції та доставки (CI/CD) реалізовані за допомогою GitLab. Репозиторій проєкту інтегровано з GitLab Runner, що дозволяє автоматично збирати, тестувати та розгортати застосунок при кожному оновленні коду. Це скорочує час релізів, мінімізує ймовірність помилок при деплої та забезпечує стабільність оновлень.

Завдяки такому підходу архітектура системи є розділеною, масштабованою та гнучкою до змін. Вона дозволяє швидко розгортати нові функції, оновлювати окремі частини застосунку без зупинки всієї системи та забезпечує високий рівень безпеки за рахунок ізоляції компонентів і централізованого управління доступом. Діаграму клієнт-серверної моделі наведено на рисунку 3.2.

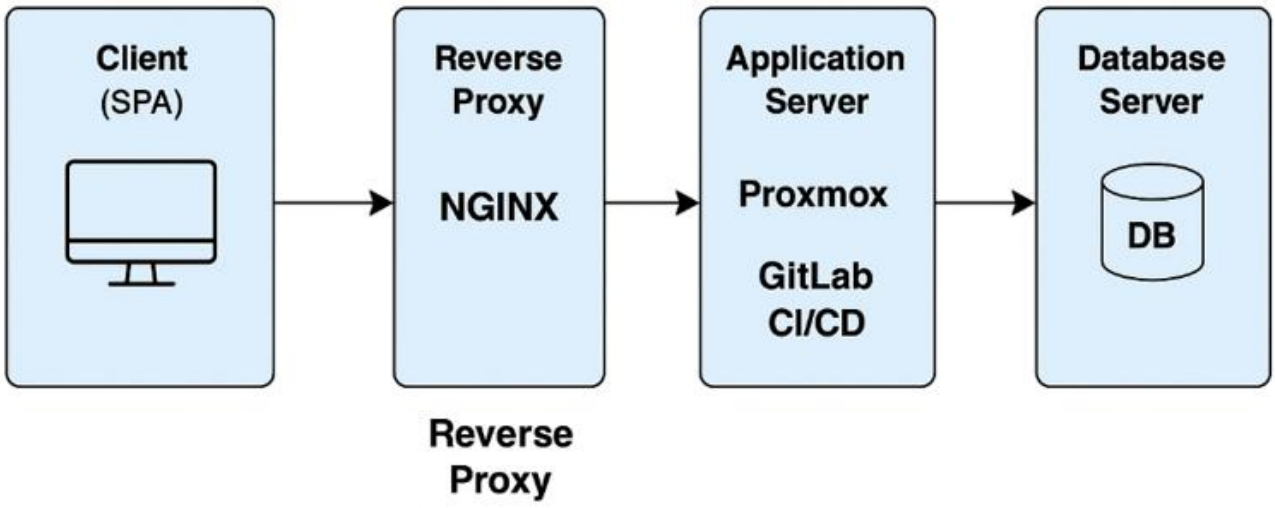


Рис. 3.2 Діаграма клієнт-серверної моделі

3.3 Архітектура клієнтської частини

Клієнтська частина web-застосунку побудована на основі сучасного JavaScript-фреймворку Vue.js, що дозволяє створювати реактивні та динамічні інтерфейси користувача. Вся логіка клієнта реалізована у вигляді односторінкового застосунку[17] (SPA), що дозволяє завантажувати лише одну HTML-сторінку і далі оновлювати лише її частини без повного перезавантаження. Це значно покращує користувацький досвід і підвищує швидкість взаємодії з інтерфейсом.

В основі архітектури клієнта лежить компонентний підхід: кожна функціональна частина інтерфейсу реалізована у вигляді окремого Vue-компонента. Компоненти мають чітко визначені обов’язки та можуть бути повторно використані в інших частинах інтерфейсу. Також застосунок використовує модульну організацію коду, що спрощує його супровід та масштабування.

Для навігації між сторінками використовується бібліотека Vue Router, яка реалізує маршрутизацію без перезавантаження сторінки.

Кожен маршрут прив’язаний до відповідного компонента, що дозволяє зберігати логічну структуру SPA та реалізувати захищені маршрути за потреби.

Уся логіка керування станом реалізована за допомогою Vuex — офіційної бібліотеки для глобального сховища даних. Основним об’єктом, який зберігається у Vuex, є таблиця з переліком технічних засобів (Dashboard). Зміни в стані автоматично відображаються в інтерфейсі, що забезпечує синхронізацію між візуальним представленням даних та фактичним станом на сервері.

Взаємодія з сервером реалізована через бібліотеку Axios, яка дозволяє виконувати асинхронні HTTP-запити. Вона забезпечує обмін даними з REST API, що розміщене на стороні Laravel, і підтримує обробку заголовків, статус-кодів, помилок тощо. Axios інтегровано у компоненти Vue та використовується як у формальних запитах (отримання, створення, оновлення даних), так і в логіці керування помилками та обробки відповіді API.

На рівні клієнта рольове розмежування не реалізовано — вся перевірка прав доступу здійснюється на стороні серверу. Проте в інтерфейсі реалізовано перевірки автентичності, які дозволяють обмежити доступ до частини функціональності, якщо користувач не авторизований. Завдяки цьому архітектура клієнтської частини є простою, гнучкою та придатною до масштабування в майбутньому.

Діаграму компонентів Vue наведено на рисунку 3.3.

Серверна частина застосунку реалізована на основі PHP-фреймворку Laravel, який обрано завдяки його сучасному підходу до розробки, широкій екосистемі та зручному API. У межах системи серверна логіка відповідає за обробку запитів від клієнтської частини, валідацію, перевірку прав доступу, взаємодію з базою даних та формування відповідей у форматі JSON[18].

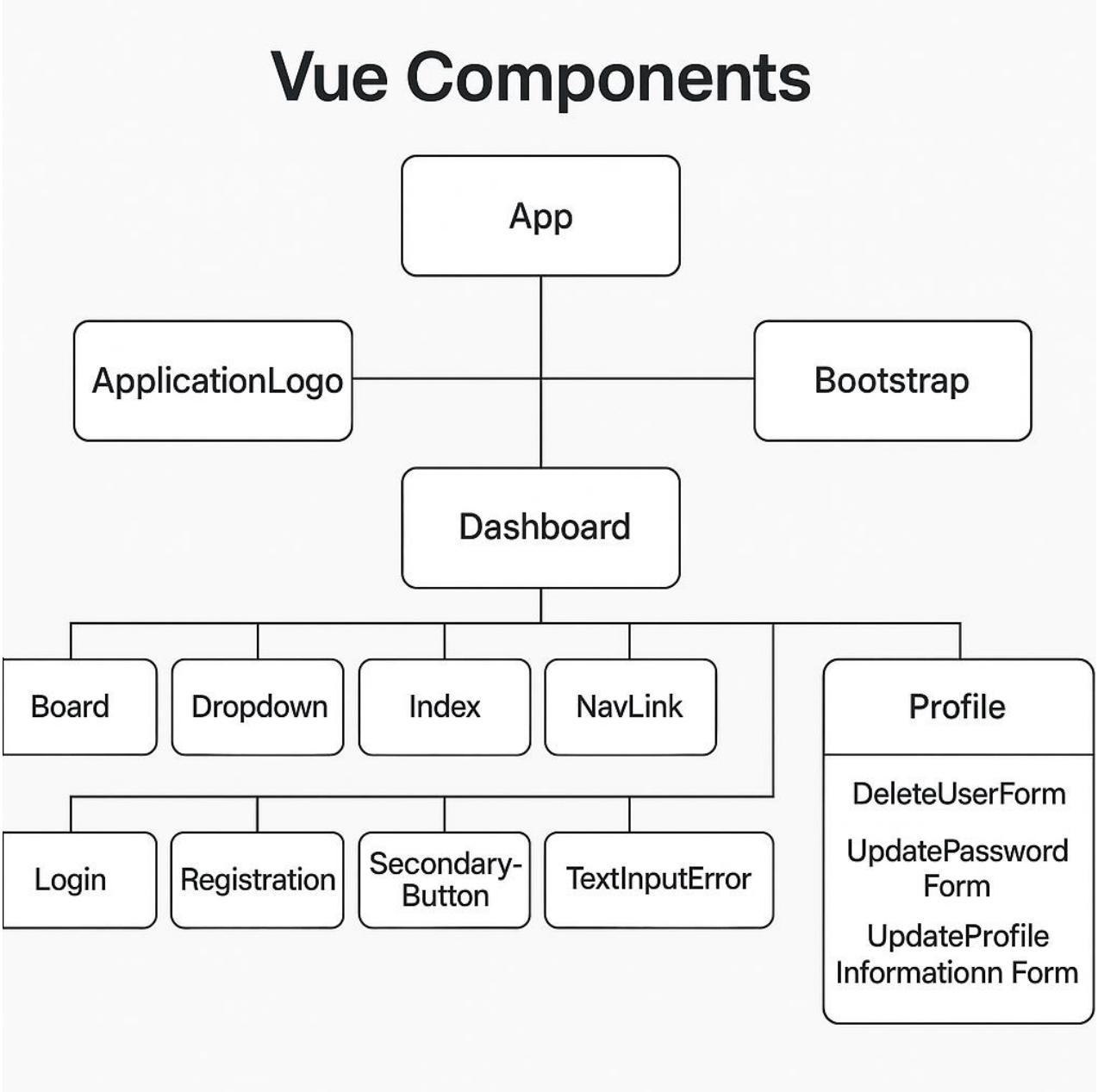


Рис. 3.3 Діаграма компонентів Vue

3.4 Архітектура серверної частини

Кожен функціональний блок реалізований у вигляді окремого контролера, який відповідає за один тип ресурсу, наприклад, користувачів, технічні засоби або кімнати. Контролери взаємодіють з моделями Eloquent ORM[19], що забезпечують об'єктно-орієнтовану роботу з базою даних, включаючи фільтрацію, сортування, зв’язки та жорстку типізацію.

Маршрутизація реалізована відповідно до принципів REST: кожен маршрут відповідає за певну дію над ресурсом (перегляд, створення, оновлення, видалення). Завдяки цьому API залишається інтуїтивно зрозумілим і зручним для інтеграції з клієнтською частиною.

Для захисту маршрутів використовуються middleware, які перевіряють наявність автентифікованого користувача. Однак повне рольове розмежування доступу на рівні серверної логіки наразі не реалізовано, але система готова до його впровадження за допомогою механізмів Laravel Gate або Policy.

Система автентифікації базується на інтеграції з Active Directory через DNS[20]. Це дозволяє забезпечити доступ до системи лише для авторизованих працівників компанії, які входять до відповідної групи користувачів в AD. Таким чином, відсутня необхідність у створенні локальних облікових записів, що знижує навантаження на адміністрування користувачів і забезпечує дотримання корпоративних стандартів безпеки.

Усі вхідні запити проходять перевірку коректності даних через вбудовані засоби валідації Laravel. Це дозволяє відхилити запити з відсутніми або некоректними параметрами ще до початку обробки логіки. Обробка винятків винесена в окремі класи, що дозволяє системі стабільно реагувати на помилки й повертати клієнту інформативні повідомлення.

Завдяки використанню Laravel серверна частина є масштабованою, розширюваною та придатною для впровадження нових функцій без ризику порушення вже реалізованих механізмів. Така архітектура дозволяє забезпечити високу продуктивність, простоту супроводу і відповідність сучасним стандартам розробки корпоративних інформаційних систем. Діаграму компонентів Laravel наведено на рисунку 3.4.

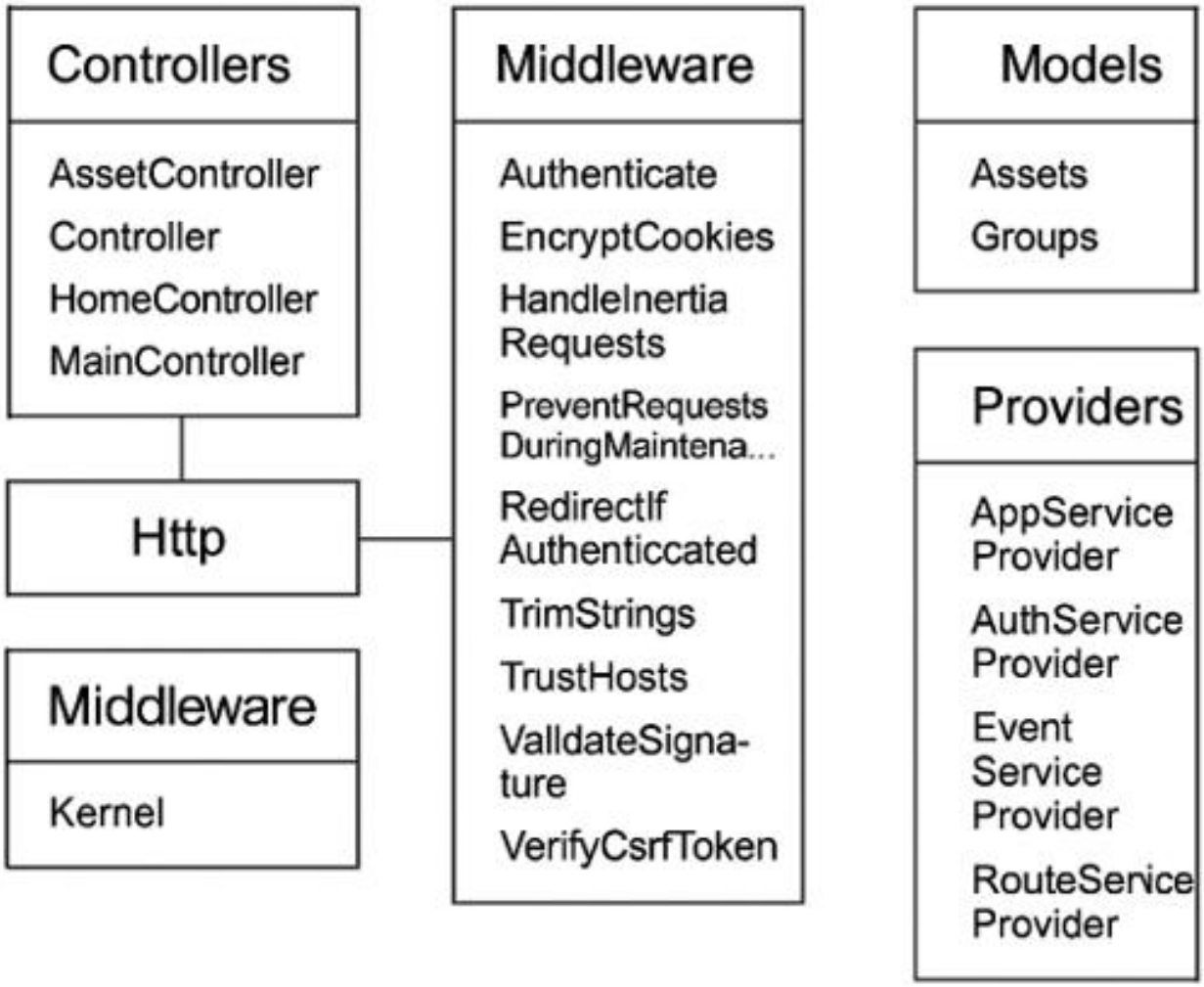


Рис. 3.4 Діаграма компонентів Laravel

3.5 Архітектура бази даних

Архітектура бази даних є фундаментальною складовою програмного забезпечення, що визначає спосіб зберігання, структурування та логічного впорядкування даних, які циркулюють у системі. У даній роботі використовується реляційна база даних MySQL, яка забезпечує надійне, структуроване та масштабоване середовище для зберігання інформації. Вибір саме MySQL обґрунтований її високою продуктивністю, стабільністю, відкритістю коду та широкою підтримкою в індустрії.

Модель бази даних побудована відповідно до вимог нормалізації — зокрема, третьої нормальної форми (3НФ), що дозволяє мінімізувати дублювання даних, уникнути аномалій оновлення та забезпечити логічну узгодженість записів. Кожна таблиця в системі виконує строго визначену функцію в загальній структурі обліку активів, користувачів, локацій, статусів та історії змін.

Основними сутностями в моделі є такі таблиці: User, Assets, Post, Groups, Room, Status. Таблиця User містить дані про працівників, які взаємодіють із системою. Assets зберігає інформацію про всі технічні засоби, включаючи інвентарні та серійні номери, дату закупівлі, статус та розміщення. Post використовується для ведення історії інвентаризації або переміщення техніки. Таблиця Groups дозволяє реалізувати організаційно-адміністративну структуру підприємства, пов’язуючи користувачів з певними підрозділами. Room відображає фізичне розташування техніки, а Status — її поточний технічний стан.

Між сутностями реалізовано типові зв’язки «один до багатьох» та «багато до одного». Наприклад, один користувач може бути відповідальним за багато записів у Assets або Post. Також кожен актив прив’язаний до певного roomId та statusId, що дозволяє оперативно визначити його місцезнаходження та технічний стан. Такі зв’язки відображають реальні відносини між об’єктами системи, що забезпечує логічну узгодженість і зручність у побудові SQL-запитів.

Хоча фізично обмеження цілісності (індекси, зовнішні ключі) не були реалізовані на рівні бази даних, усі обмеження та перевірки закладено на рівні серверної логіки у Laravel. Це дозволяє більш гнучко керувати даними та залишає можливість масштабування структури без порушення існуючих зв’язків.

Окремим аспектом архітектури є реалізація логування змін. Зокрема, при кожному оновленні або видаленні активу відповідні зміни реєструються в таблиці Post. Це дозволяє відслідковувати історію переміщень технічних засобів, контролювати відповідальних осіб та забезпечувати прозорість дій у системі. Такий підхід є важливим з погляду аудиту, внутрішнього контролю та

інформаційної безпеки.

Таким чином, архітектура бази даних забезпечує комплексну підтримку основних процесів системи — облік, контроль, інвентаризацію, аналітику, а також дозволяє масштабувати систему в майбутньому, додаючи нові функціональні блоки без кардинальної зміни її структури. Діаграму класів бази даних наведено на рисунку 3.5.

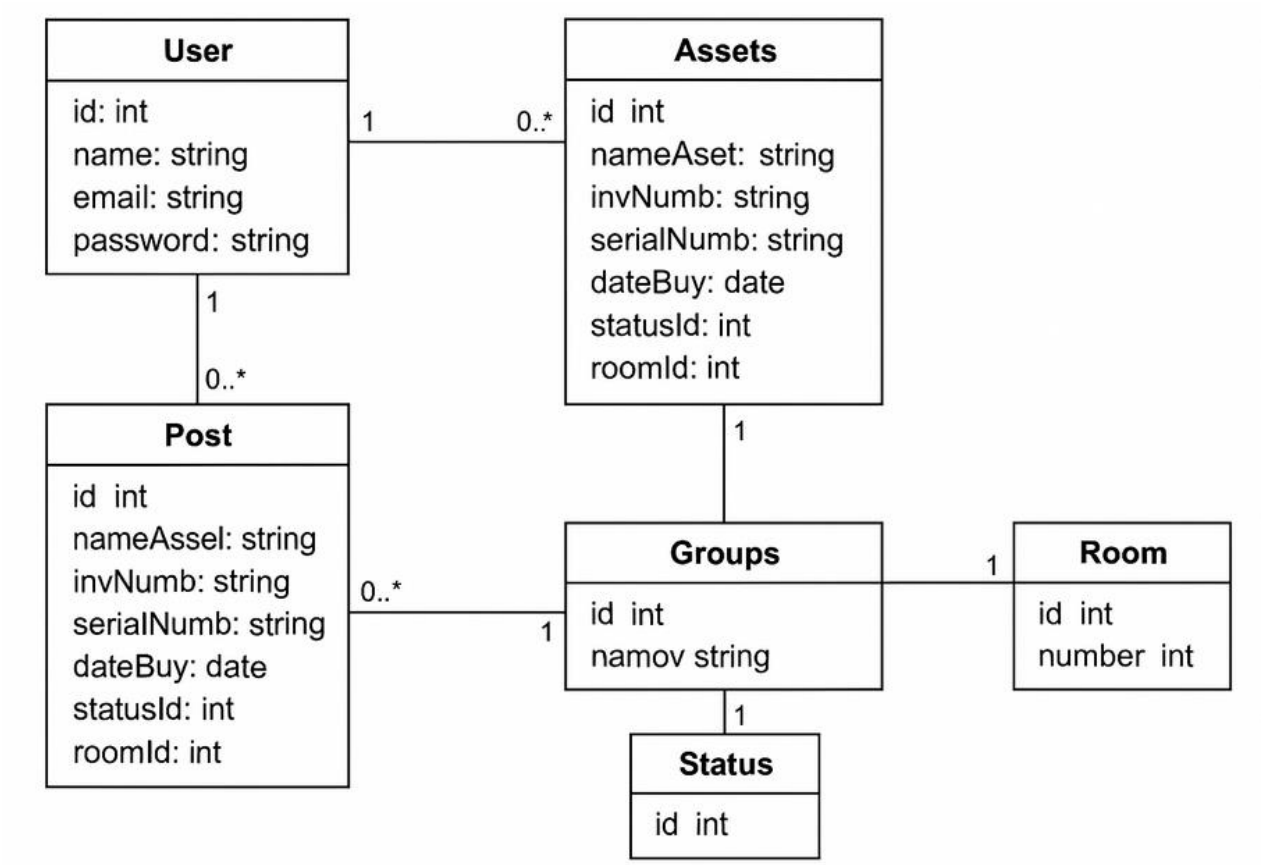


Рис. 3.5 Діаграма класів бази даних

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Дизайн інтерфейсу

Дизайн інтерфейсу користувача є важливою складовою зручності та ефективності використання системи. У межах цієї роботи було реалізовано класичний та мінімалістичний підхід до побудови інтерфейсу. Основна увага зосереджена на практичності, швидкому доступі до функцій та легкості сприйняття інформації.

На верхній панелі навігації розташовані основні розділи системи: «Board», «Rooms», «Serial numbers», «Employees». Кожна з вкладок веде на відповідну сторінку зі своїм функціональним призначенням. Наприклад, «Board» — це головна таблиця обліку техніки; «Rooms» — список приміщень, до яких можна прикріпити активи; «Employees» — список відповідальних осіб.

Основна частина інтерфейсу зосереджена на таблиці активів, де виводиться перелік технічних засобів із зазначенням їх інвентарного та серійного номера, статусу, кімнати, до якої вони закріплені, а також можливістю керування цими записами. Кожен рядок містить чекбокс для вибору одного або декількох записів, що дає змогу здійснювати групові операції (наприклад, переміщення в іншу кімнату або зміну відповідального).

Інтерфейс розміщення елементів продуманий таким чином, щоб забезпечити логічне групування керуючих елементів. У верхній частині таблиці розташовані кнопки дій: «Створити пост», «Створити групу», «Призначити відповідального», «Перемістити в кімнату», «Перемістити в групу», а також поле для пошуку. Кнопки мають кольорову ідентифікацію для зручності: зелені — для створення та редагування, сині — для перегляду, сірі — неактивні або службові дії.

Під таблицею реалізовано пагінацію — користувач має можливість переглядати дані порціями, переходити до попередньої або наступної сторінки.

Колірна схема інтерфейсу нейтральна, фокус зроблено на доступності та контрастності: фонові елементи світло-сірі, активні кнопки — сині або зелені, текст — темно-сірий або чорний. Для виділення окремих рядків або груп даних використовуються альтернативні кольори фону (наприклад, жовтий для виділення або сірий для підгруп).

Інтерфейс не використовує готових UI-бібліотек (на кшталт Bootstrap або Tailwind CSS), всі стилі створені вручну, що дозволяє гнучко керувати зовнішнім виглядом і уникати надлишкових залежностей. Завдяки цьому дизайн залишається легким, адаптивним і добре масштабованим під різні розміри екрану.

Таким чином, дизайн інтерфейсу є ергономічним, функціональним та інтуїтивно зрозумілим для користувача будь-якого рівня підготовки. Це дозволяє суттєво зменшити час навчання персоналу, прискорює щоденну роботу в системі та забезпечує ефективне управління технічними ресурсами. Кольорову палітру web-додатку наведено на рисунку 4.1.

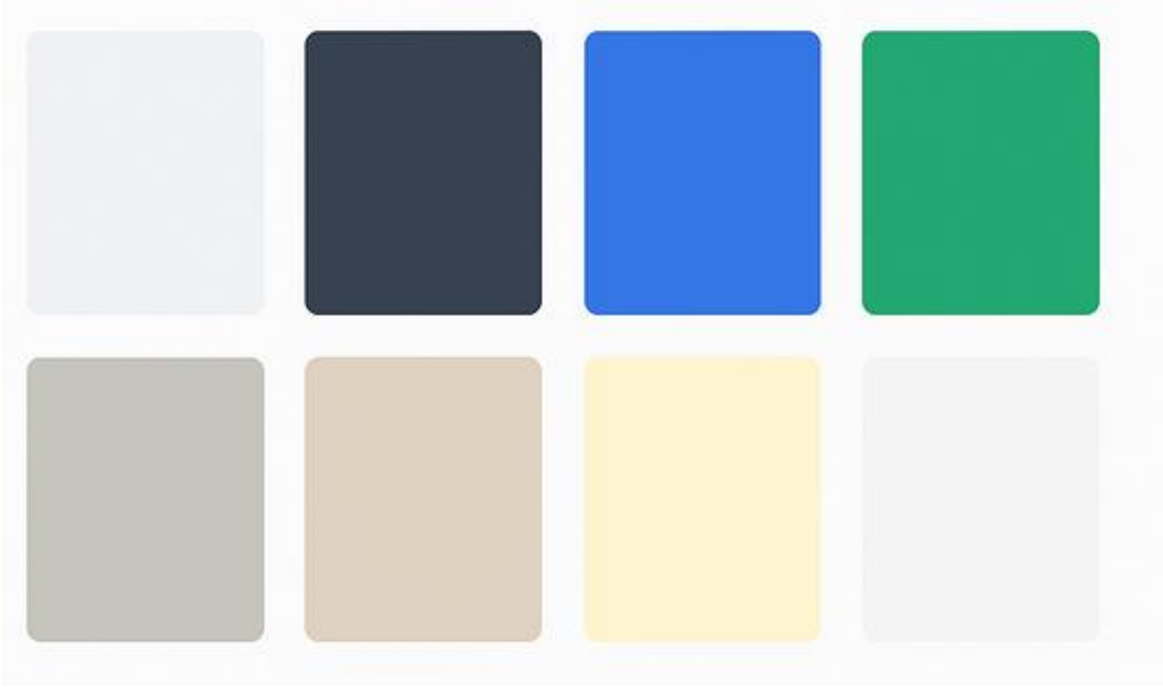


Рис. 4.1 Кольорова палітра web-додатку

4.2 Реалізація клієнтської частини

Клієнтська частина інформаційної системи реалізована як односторінковий web-застосунок (SPA) за допомогою фреймворку Vue.js, що дозволяє забезпечити сучасну взаємодію з користувачем без повного перезавантаження сторінок. Уся логіка та структура фронтенду побудовані за компонентним принципом, що дає змогу ізолювати функціональні блоки, повторно їх використовувати та легко масштабувати систему. Список компонентів Vue наведено на рисунку 4.2.

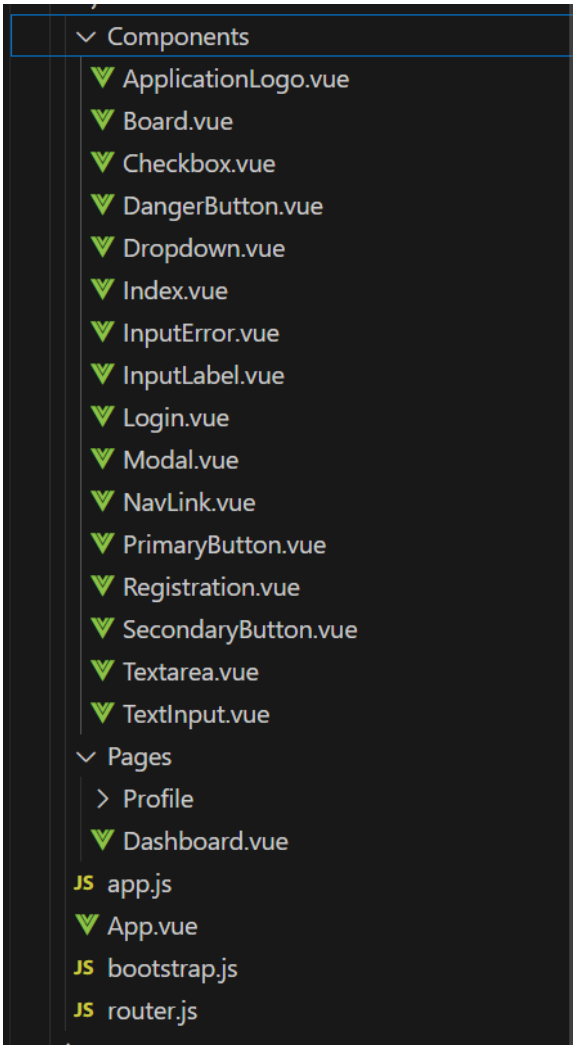


Рис. 4.2 Список компонентів Vue

Навігація між розділами застосунку реалізована через Vue Router, що забезпечує динамічну маршрутизацію без зміни основної HTML-структури. Основними сторінками є: Dashboard (таблиця активів), Rooms (облік приміщень), Serial numbers (перелік техніки за серійними номерами), Employees (перелік відповідальних осіб). Кожна сторінка має окремий Vue-компонент, у межах якого реалізовано специфічну логіку — наприклад, отримання даних із сервера через Axios, обробку користувацьких дій, рендеринг таблиць тощо. Екранну форму навігації наведено на рисунку 4.3.

Рис. 4.3 Екранна форма навігації

Користувач заходить у систему через кнопку Login. Після введення логіну та пароля відбувається перевірка даних у Active Directory (через серверну частину), і в разі успішної авторизації користувач отримує доступ до всіх основних функцій. Інтерфейс динамічно перебудовується відповідно до авторизованого статусу — доступні кнопки для створення, редагування, призначення відповідальних, фільтрації та пошуку. Екранну форму авторизації наведено на рисунку 4.4.

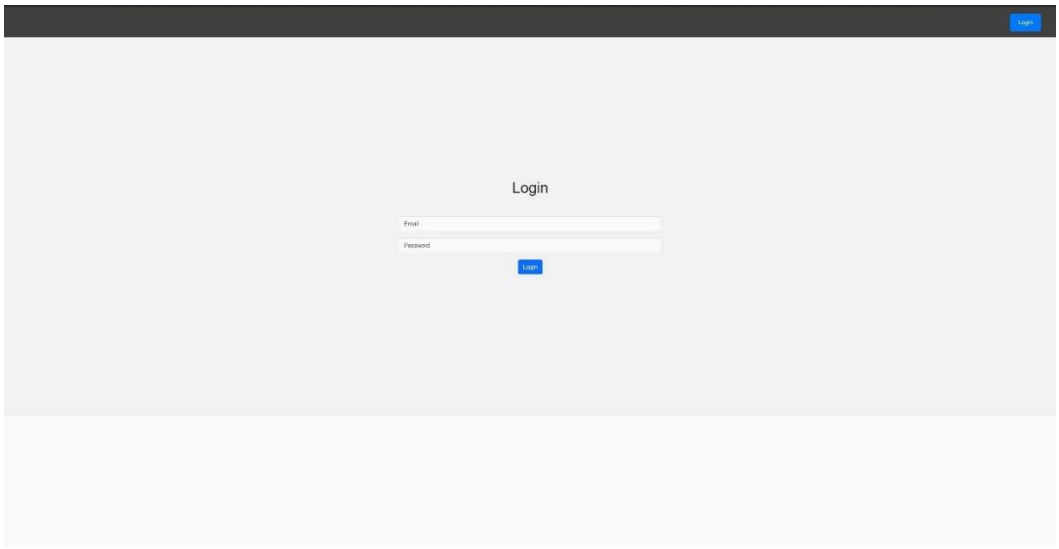


Рис. 4.4 Екранна форма авторизації користувача

Створення нового запису в системі реалізовано у вигляді модального вікна або окремої форми, яка з’являється після натискання кнопки «Створити пост». Користувач заповнює всі необхідні поля (назва активу, інвентарний номер, серійний номер, кімната, статус), після чого ініціюється HTTP POST-запит до серверної частини. Усі поля проходять клієнтську валідацію: перевіряється, чи заповнені обов’язкові значення, правильність формату, допустимість введених значень тощо. Екранну форму створення посту наведено на рисунку 4.5.

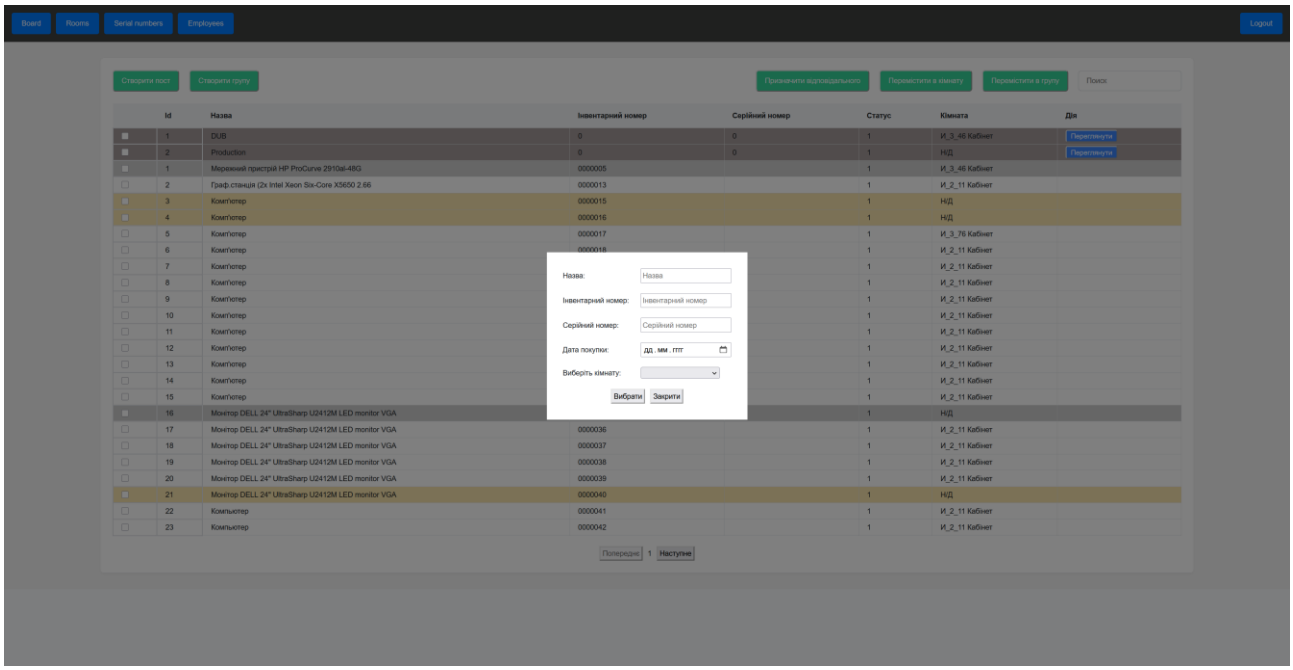


Рис. 4.5 Екранна форма авторизації користувача

Стан основної таблиці Dashboard зберігається та керується через Vuex — глобальне сховище даних, інтегроване в застосунок. Vuex дозволяє централізовано обробляти всі зміни, які відбуваються в інтерфейсі. Наприклад, після успішного створення нового активу таблиця оновлюється автоматично — система або отримує оновлені дані з API, або додає новий запис безпосередньо у сховище. Такий підхід дозволяє мінімізувати кількість повторних запитів до сервера, покращити продуктивність і забезпечити стабільність роботи застосунку при великій кількості даних. Екранну форму Dashboard наведено на рисунку 4.6.

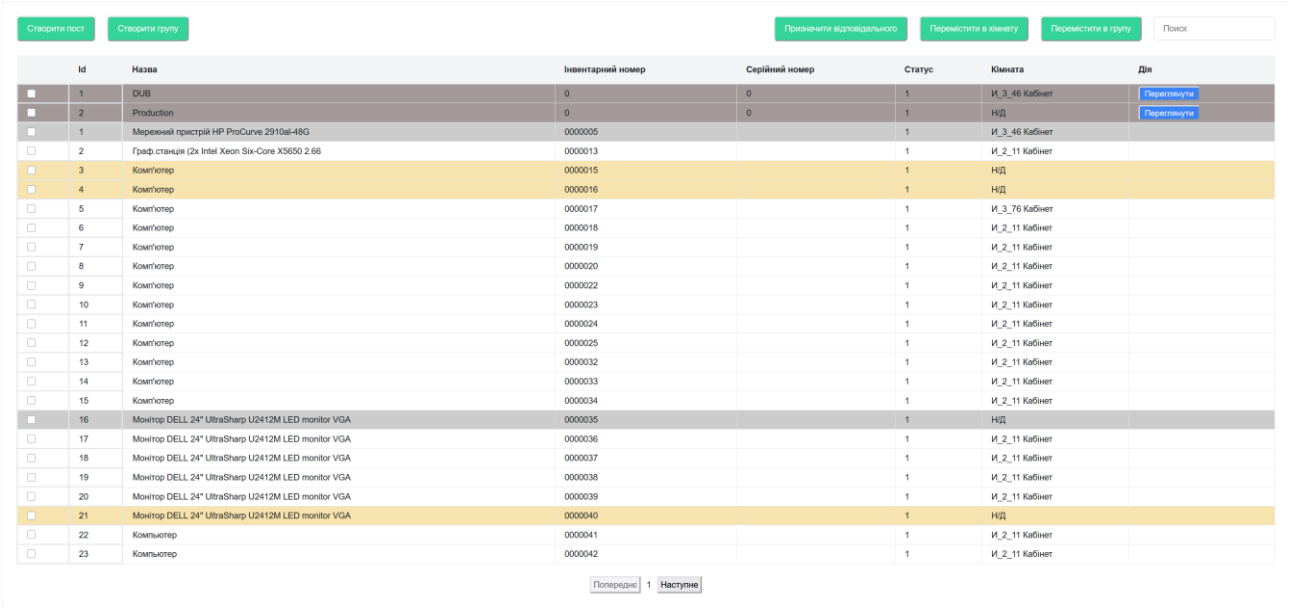


Рис. 4.6 Екранна форма Dashboard

Комунікація між клієнтською та серверною частинами відбувається через Axios — бібліотеку для виконання асинхронних HTTP-запитів. Вона забезпечує повну підтримку REST API, роботу з токенами автентифікації, перехоплення запитів і обробку помилок. Axios інтегровано в усі ключові компоненти Vue: через неї реалізовано завантаження списків, створення, оновлення та видалення записів, а також обробку серверних відповідей. Приклад Axios запиту наведено на рисунку 4.7.

```
const actions = {
  fetchItems({ commit }) {
    axios.get('http://localhost:8000/assets/getall')
      .then(response => {
        commit('setItems', response.data);
      })
      .catch(error => {
        console.log(error);
      });
  },
  setPage({ commit }, page) {
    commit('setPage', page);
  },
};
```

Рис. 4.7 Приклад Axios запиту

У системі реалізовано базову обробку помилок на стороні клієнта. Якщо під час запиту до сервера виникає помилка — наприклад, через валідацію або недоступність API — інтерфейс показує відповідне повідомлення користувачу. Це дозволяє уникати неочікуваних збоїв у роботі та забезпечує інформативний зворотний зв’язок. Приклад некоректно вказаних даних наведено на рисунку 4.8.

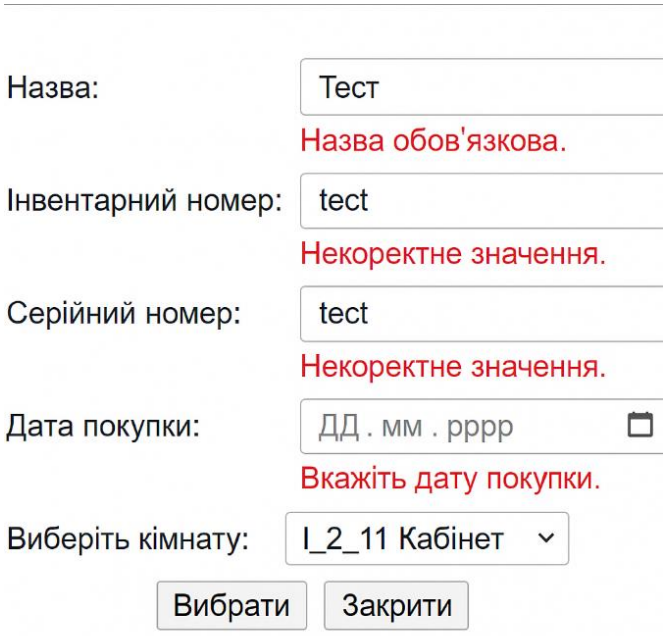


Рис. 4.8 Приклад некоректно вказаних даних

Клієнтська частина не реалізує повноцінне рольове керування доступом — усі перевірки здійснюються на рівні серверної логіки. Проте клієнт адаптується до статусу користувача та не показує функціональність, яка не повинна бути доступною неавторизованим користувачам. Це реалізовано через перевірку автентифікаційного статусу, який зберігається у сховищі (Vuex або cookies).

Загалом клієнтська частина побудована за сучасними стандартами розробки SPA: ізоляція компонентів, реактивність, централізоване зберігання стану, асинхронна взаємодія з сервером, динамічна маршрутизація та базовий рівень безпеки. Така структура забезпечує масштабованість, стабільність та зручність користування навіть у великих об’ємах даних.

4.3 Реалізація серверної частини

Серверна частина web-застосунку реалізована з використанням PHP-фреймворку Laravel, що дозволяє ефективно реалізовувати логіку взаємодії з клієнтською частиною, керування даними та забезпечення захисту на рівні бекенду. Всі запити від фронтенду надходять до API-сегменту Laravel, який обробляє їх відповідними контролерами.

У структурі контролерів виділяються: AssetController, який відповідає за обробку CRUD-операцій над технічними засобами; MainController, який обробляє навігаційні маршрути для SPA; GetController, який реалізує логіку отримання окремих даних; HomeController, що може використовуватись як точка входу після авторизації. Приклад контролера AssetController наведено на рисунку 4.9.

```php
class AssetController extends Controller
{
    //
    public function getAll()
    {
        $allAssets = Assets::select('id','name','invNumb','serialNumb','statusId','roomId')->where('inGroup', '=', '0')->get()->toArray();
        $allGroups = Groups::select('id','name','invNumb','serialNumb','statusId','roomId')->get()->toArray();

        $toAnswer = array_merge( $allGroups, $allAssets);

        return response()->json($toAnswer);
    }

    public function getDefault()
    {
        $tu = [
            [
                'id' => '1',
                'nameAsset' => 'Клавіатура',
                'invNumb' => '000956324',
                'dateBuy' => '01-02-2023',
                'statusId' => '1',
                'roomId' => '1',
            ],
            [
                'id' => '2',
                'nameAsset' => 'Мишка',
                'invNumb' => '00324366',
                'dateBuy' => '01-02-2022',
                'statusId' => '1',
                'roomId' => '2',
            ],
        ];

        return response()->json($tu);
    }
}
```

Рис. 4.9 Приклад контролера AssetController

Маршрутизація реалізована відповідно до REST-підходу. Усі API-запити згруповані з використанням префіксів, що спрощує підтримку коду. Наприклад, для активів створено окрему групу маршрутів під префіксом /assets, у якій визначено методи для отримання, додавання та видалення записів. Кожен маршрут асоційований із відповідним методом у контролері, що забезпечує логічну зв'язність між URL, дією та результатом. Приклад API-запитів Assets наведено на рисунку 4.10.

```
Route::group(['namespace' => 'Assets', 'prefix' => 'assets'], function () {
    Route::get('/getall', [ AssetController::class, 'getAll']);
    Route::get('/add', [ AssetController::class, 'addAsset']);
    Route::get('/delete', [ AssetController::class, 'delAsset']);
});
```

Рис. 4.10 Приклад API-запитів Assets

Для реалізації автентифікації використовується інтеграція з Active Directory (AD). Після введення логіну й пароля користувача сервер звертається до AD для перевірки достовірності облікових даних. Якщо перевірка проходить успішно, користувач отримує CSRF-токен, який є дійсним протягом 12 годин. Це забезпечує захист від підроблених запитів та дозволяє підтримувати сесію з клієнтом у безпечному режимі. Вхід у систему обмежений лише для користувачів, які належать до визначеної AD-групи, що гарантує відповідність політикам внутрішньої безпеки. Надання групи доступу в Active Directory до web-додатку наведено на рисунку 4.11.

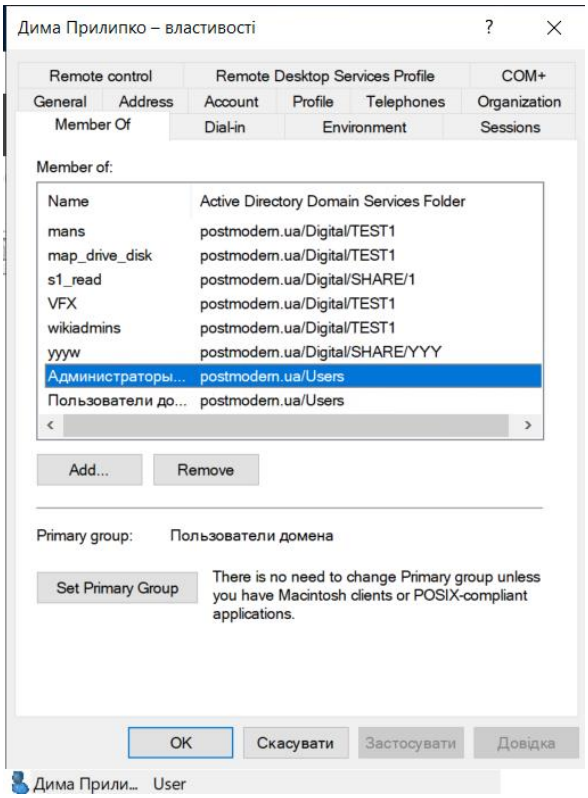


Рис. 4.11 Надання групи доступу в Active Directory до web-додатку

Для захисту маршрутів Laravel використовує middleware. Усі запити до приватних API проходять перевірку через шар автентифікації, що унеможливлює несанкціонований доступ до ресурсів. Крім того, вбудовані засоби валідації Laravel використовуються для перевірки полів у запитах. Наприклад, при створенні нового запису перевіряється коректність інвентарного номера, унікальність запису, правильний формат серійного номера та інші параметри. Це дозволяє попередити помилки ще до збереження інформації в базі даних. Приклад класу перевірки аутентифікації користувача наведено на рисунку 4.12.

```php
class RedirectIfAuthenticated
{
    public function handle(Request $request, Closure $next, string ...$guards): Response
    {
        $guards = empty($guards) ? [null] : $guards;

        foreach ($guards as $guard) {
            if (Auth::guard($guard)->check()) {
                return redirect(RouteServiceProvider::HOME);
            }
        }

        return $next($request);
    }
}
```

Рис. 4.12 Приклад класу перевірки аутентифікації користувача

Усі основні дані моделюються за допомогою Eloquent ORM[16]. Кожна сутність у системі має відповідну модель: Assets, Post, User, Groups, Room. Зв’язки між моделями реалізовані згідно з логікою відношень у базі даних (один-до-багатьох, багато-до-одного), що дозволяє ефективно вибирати, фільтрувати та оновлювати пов’язані дані. Приклад моделі Post наведено на рисунку 4.13.

```php
class Post extends Model
{
    use HasFactory;

    protected $fillable = [
        'nameAsset', 'invNumb' ,'title', 'body',
    ];
}
```

Рис. 4.13 Приклад моделі Post

Таким чином, серверна частина забезпечує не лише функціональну обробку запитів, а й виконує ключові завдання з безпеки, валідації, зберігання й представлення даних у зручному форматі для клієнтської частини. Завдяки розподілу логіки на модулі та контролери, система є зручною в підтримці, розширенні й інтеграції з іншими сервісами. Діаграму активності створення Post та діаграму активності призначення відповідального наведено на рисунках 4.14 та 4.15.

Рис. 4.14 Діаграма активності створення Post

Рис. 4.15 Діаграма активності призначення відповідального

4.4 Завантаження роботи на GitLab та проходження CI/CD

Для забезпечення контрольованої командної розробки, автоматизованого тестування та зручного розгортання застосунку було обрано систему керування версіями Git у поєднанні з платформою GitLab. Уся кодова база зберігається у приватному GitLab-репозиторії з обмеженим доступом, що відповідає вимогам безпеки та дозволяє точно відслідковувати історію змін, проводити рев'ю коду, вести облік задач та зручно обговорювати оновлення.

В основі процесу автоматизації CI/CD (Continuous Integration / Continuous Deployment) лежить файл конфігурації .gitlab-ci.yml, який описує всі етапи: перевірка коду (lint), збирання фронтенду, виконання команд Laravel (наприклад, php artisan migrate, config:cache, route:cache), тестування та деплой на сервер. Приклад коду .gitlab-ci.yml наведено на рисунку 4.16.

```yaml
stages:
  - lint
  - test
  - build_frontend
  - laravel
  - deploy

variables:
  NODE_ENV: production
  DEPLOY_USER: deploy_user
  DEPLOY_SERVER: 123.123.123.123
  DEPLOY_PATH: /var/www/project

cache:
  key: ${CI_COMMIT_REF_SLUG}
  paths:
    - node_modules/
    - vendor/

before_script:
  - echo "🏗️ Початкова підготовка..."
  - apt-get update -yqq && apt-get install -yqq git curl unzip
  - cp .env.ci .env
  - composer install --no-interaction --prefer-dist
  - npm ci

lint_code:
  stage: lint
  script:
    - echo "🔍 Lint PHP"
    - ./vendor/bin/pint --test
    - echo "🔍 Lint JS"
    - npm run lint
  only:
    - main

run_tests:
  stage: test
  script:
    - echo "🧪 Тестування"
    - php artisan config:clear
    - php artisan migrate --env=testing
    - ./vendor/bin/phpunit
  only:
    - main

build_frontend:
  stage: build_frontend
  script:
    - echo "🛠️ Збирання фронтенду"
    - npm run build
  artifacts:
    paths:
      - public/build
  only:
    - main

prepare_laravel:
  stage: laravel
  script:
    - echo "⚙️ Laravel підготовка"
    - php artisan config:cache
    - php artisan route:cache
    - php artisan view:cache
  only:
    - main
```

Рис. 4.16 Приклад коду .gitlab-ci.yml

Після кожного пушу в основну гілку GitLab автоматично ініціює CI-процес: збирається інтерфейс, перевіряється коректність синтаксису JavaScript/PHP-файлів, виконується базовий юніт-тест або симуляція запуску застосунку. Якщо всі етапи проходять успішно, система переходить до розгортання — застосунок деплоїться на сервер, що працює у віртуалізованому середовищі Proxmox. Результат проходження CI-процесу наведено на рисунку 4.17.

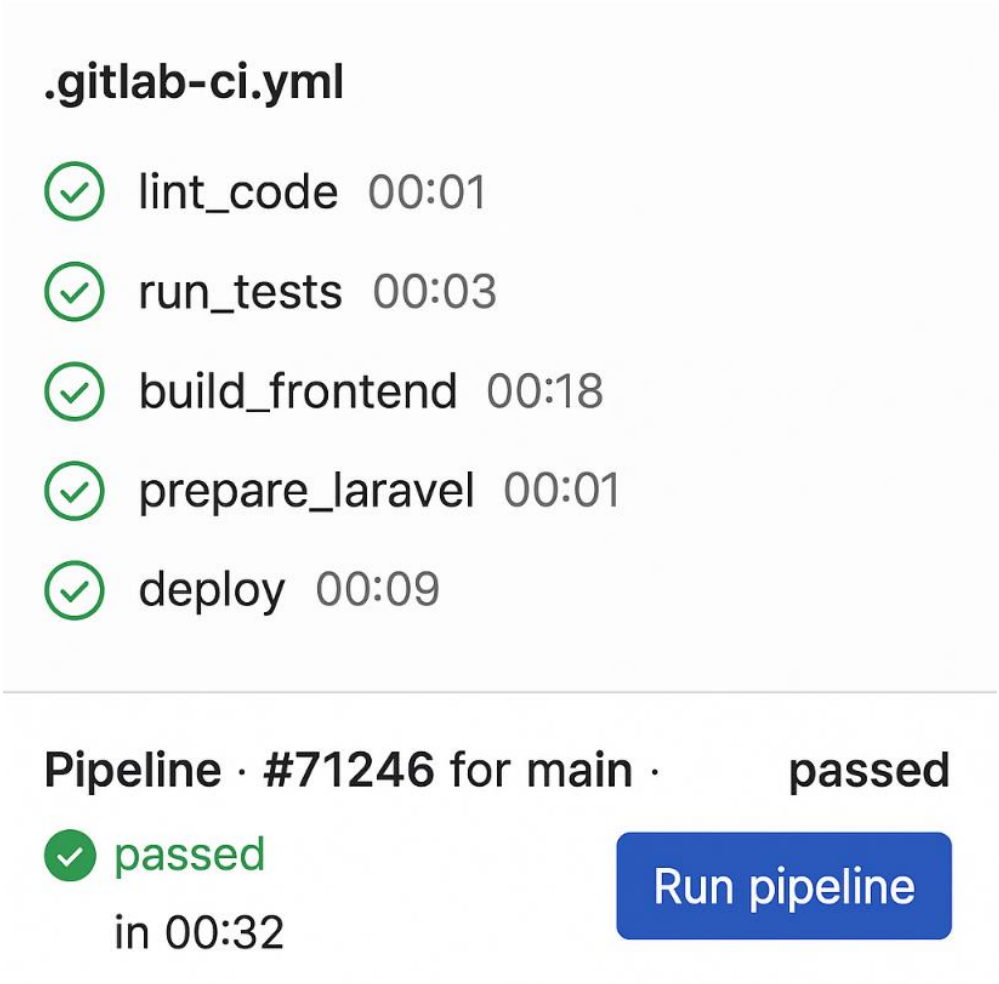


Рис. 4.17 Результат проходження CI-процесу

Proxmox використовується як платформа для запуску ізольованих середовищ — кожна версія застосунку може бути розгорнута у власній віртуальній машині або контейнері. Завдяки цьому забезпечується відокремлення робочих середовищ (dev, staging, production) та гнучке керування ресурсами. Автоматичне розгортання включає копіювання оновленого коду, перезапуск служб, очищення кешу та перевірку працездатності. Екранну форма віртуальних середовищ Proxmox наведено на рисунку 4.18.

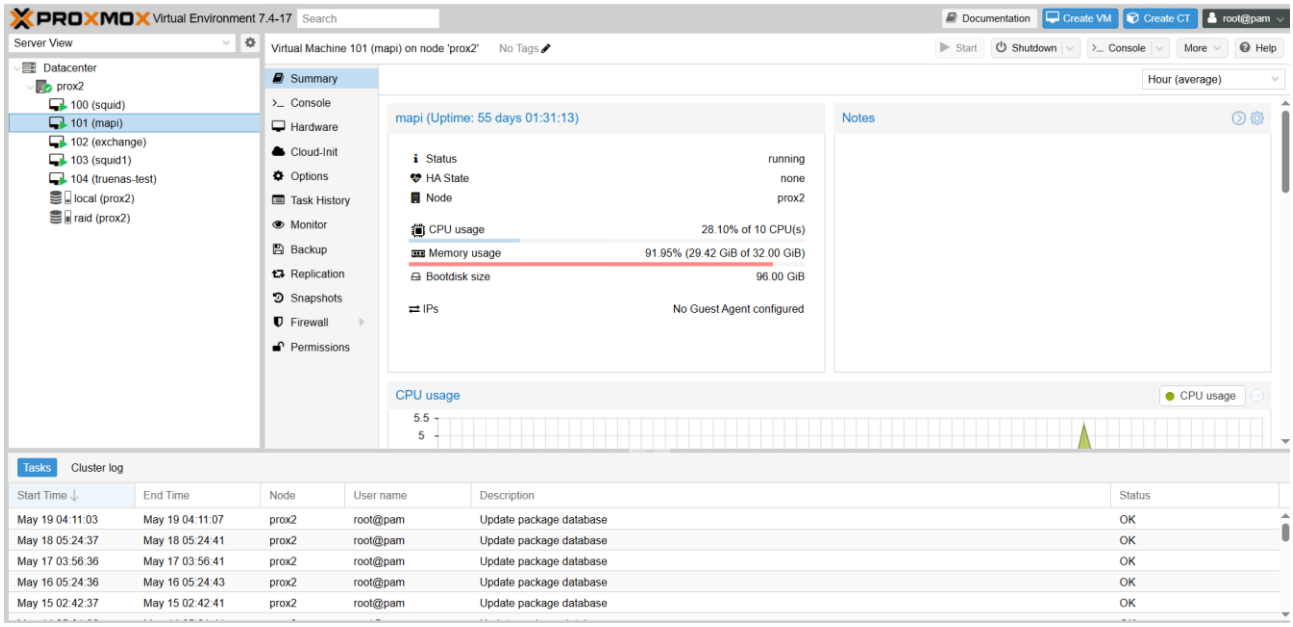


Рис. 4.18 Екранна форма віртуальних середовищ Proxmox

Переваги такого підходу — це стабільність процесу розгортання, зменшення ризику людських помилок, підвищення швидкості доставки оновлень і прозорість усіх змін. Завдяки впровадженню CI/CD розробка застосунку стала безперервною, контрольованою та готовою до масштабування. Діаграму впровадження наведено на рисунку 4.19.

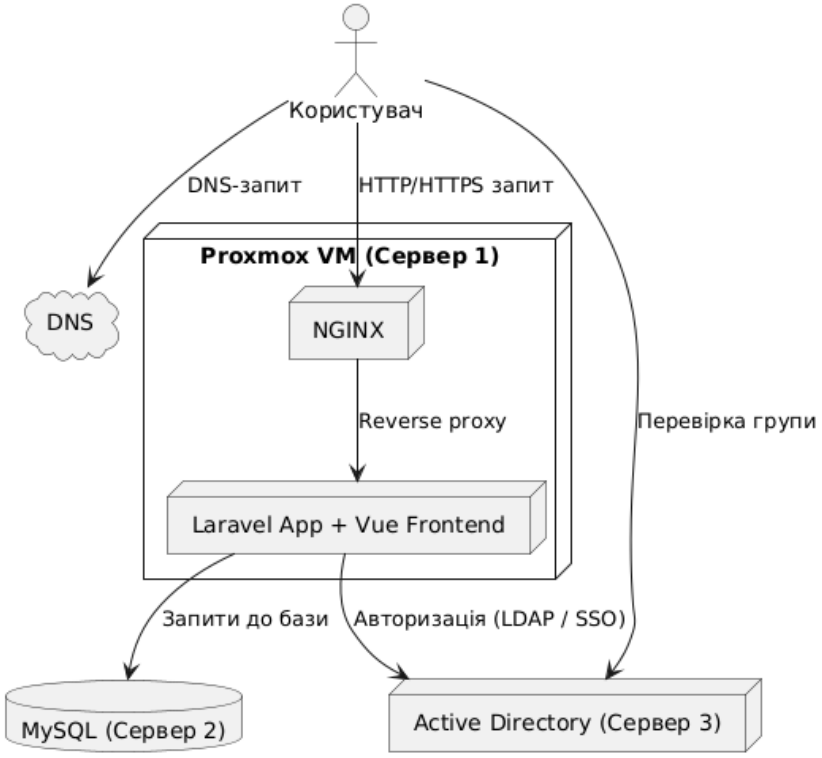


Рис. 4.19 Діаграма впровадження

4.5 Тестування застосунку

Тестування застосунку є невід'ємним етапом розробки, що дозволяє виявити недоліки, перевірити відповідність функціональності вимогам та гарантувати стабільну роботу системи в різних умовах. У межах даної роботи тестування здійснювалося вручну, із залученням функціонального тестування, регресійного перегляду та сценаріїв на основі реального використання.

У процесі тестування були перевірені всі ключові модулі системи: інтерфейс авторизації (включно з перевіркою автентифікації через Active Directory), створення, редагування, перегляд і видалення постів (технічних засобів), взаємодія з кімнатами та групами, пошук та фільтрація, обробка помилок у валідації та стани взаємодії користувача з інтерфейсом. Окрема увага приділялася стабільності роботи таблиці Dashboard при великій кількості записів, а також коректності оновлення даних у реальному часі через Vuex.

Протягом перевірки було зафіксовано декілька критичних і незначних

помилок, серед яких:

- некоректне оновлення статусу активу при зміні відповідального;
- дублювання інвентарного номера при повторному натисканні кнопки «Створити» без очікування відповіді серверу;
- неправильне округлення сторінок при пагінації, що спричиняло недоступність останніх записів у списку;
- відсутність повідомлення при помилці автентифікації користувача.

Після виявлення проблем було проведено виправлення коду на фронтенді та сервері. Кожна помилка проходила повторне тестування для впевненості у її повному усуненні. Крім того, перед фінальним деплоєм проводилось регресійне тестування, яке охоплювало всі функціональні частини системи, щоб переконатися у відсутності побічних ефектів від змін. Зведені результати тест-кейсів наведено у таблиці 4.1.

Таблиця 4.1

Зведені результати тест-кейсів web-застосунку для інтеграції процесів інвентаризації та управління технічними засобами

Назва тесту	Кроки тестування	Очікуваний результат	Статус
Авторизація користувача	Ввести логін і пароль → натиснути "Login"	Користувач успішно авторизується, відкривається інтерфейс	Пройдено
Створення нового посту	Натиснути "Створити пост" → заповнити поля → натиснути "Зберегти"	Запис додається до таблиці, повідомлення про успіх	Пройдено
Помилка валідації при створенні посту	Спробувати створити пост з порожніми полями	Показується повідомлення про помилку, запис не створюється	Пройдено

Продовження таблиці 4.1

Фільтрація за статусом	Обрати статус у фільтрі → переглянути таблицю	Виводяться лише записи з обраним статусом	Пройдено
Переміщення активу в іншу кімнату	Вибрати актив → натиснути "Перемістити в кімнату" → вибрати нову кімнату	Значення поля "Кімната" оновлюється	Пройдено
Перевірка пагінації	Прокрутити таблицю → перейти до наступної сторінки	Виводяться нові дані, нумерація працює коректно	Пройдено
Повторне натискання кнопки "Створити"	Натискати "Створити" декілька разів підряд без очікування	Створюється лише один запис, система блокує повторні запити	Пройдено
Відмова авторизації для невалідного AD	Ввести некоректний логін/пароль	Виводиться повідомлення про помилку автентифікації	Пройдено
Оновлення Dashboard після зміни активу	Змінити дані активу → зберегти	Таблиця оновлюється автоматично без перезавантаження сторінки	Пройдено
Валідація серійного номера	Ввести неправильний формат серійного номера	Виводиться помилка валідації, дані не надсилаються	Пройдено
Масове призначення відповідального	Вибрати кілька активів → натиснути «Призначити відповідального» → обрати користувача	Обраний користувач встановлюється як відповідальний для всіх вибраних записів	Пройдено

Крім того, було перевірено нефункціональні вимоги:

1 Захист персональних даних користувачів. Було підтверджено шифрування паролів за допомогою bcrypt. При спробі доступу до захищених маршрутів без токена авторизації користувач отримує повідомлення про помилку. Шифровані паролі облікового запису наведено на рисунку 4.20.

Рис. 4.20 Шифровані паролі облікового запису

2 Оптимізація запитів до бази даних. Система успішно працювала з базою, що містила понад 10 000 записів, без критичного зниження швидкодії. Елементи таблиці підвантажуються сторінково, запити обмежені та оптимізовані через Eloquent.

3 Можливість розширення функціоналу. Було реалізовано додавання нової функції без зміни існуючого коду ядра, що підтвердило модульну архітектуру.

4 Час відгуку системи. Виміряно середній час відповіді при стандартних операціях (авторизація, редагування запису, фільтрація) — у межах 0.3–1 секунд, що відповідає вимогам продуктивності. Час відгуку наведено на рисунку 4.21.

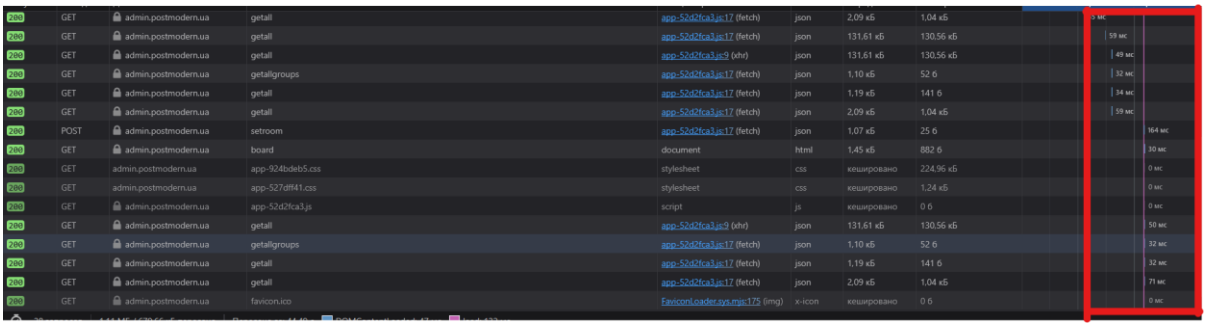


Рис. 4.21 Час відгуку

4.6 Аналіз ефективності впровадженого рішення

Після завершення розробки та тестування системи було проведено аналіз ефективності впровадженого рішення з позиції зручності користування, продуктивності, точності обліку та відповідності поставленим завданням. Основною метою розробки було створення інструменту для централізованого обліку технічних засобів, спрощення процесу інвентаризації та підвищення прозорості у взаємодії між структурними підрозділами.

У результаті впровадження системи користувачі отримали зручний інтерфейс для перегляду, редагування та групової обробки даних про обладнання. Функціональність призначення відповідального, переміщення в іншу кімнату, групу або зміну статусу дозволила автоматизувати процеси, які раніше виконувались вручну або велися у вигляді таблиць. Це дало змогу зменшити кількість помилок, пов’язаних із дублюванням інформації або її втратою.

Завдяки автоматичному збереженню історії змін через таблицю журналу дій система дозволяє проводити аудит відповідальності та контролювати повний життєвий цикл технічного засобу. Інтеграція з Active Directory забезпечила зручність і безпеку — користувачі входять до системи під своїми корпоративними обліковими записами, що зменшує адміністративне навантаження та відповідає політиці інформаційної безпеки компанії.

Ключовим показником ефективності стала швидкість доступу до даних: фільтрація, пошук, сортування та масові дії виконуються практично миттєво завдяки реактивному інтерфейсу на Vue.js у поєднанні з Vuex. Запити до бази даних оптимізовані й повертають результат із затримкою в межах 1–2 секунд навіть при великому обсязі записів.

У загальному підсумку впроваджене рішення дозволило:

- скоротити час проведення інвентаризації більш ніж на 50%;
- зменшити кількість помилок у звітності за технічними засобами;
- забезпечити надійне та структуроване зберігання інформації;

- підвищити контроль над рухом обладнання та відповідальними особами;
- адаптувати систему до специфіки компанії без потреби залучення зовнішніх сервісів.

Таким чином, впроваджений web-застосунок підтвердив свою доцільність, зручність і ефективність у реальних умовах використання в корпоративному середовищі.

4.7 Надання рекомендацій щодо подальшого розвитку

Незважаючи на ефективність поточної реалізації, система має значний потенціал для подальшого розвитку та розширення функціональності відповідно до потреб компанії та сучасних тенденцій у сфері IT-інфраструктури. Після аналізу існуючої структури, функціоналу та зворотного зв’язку від тестування було сформовано ряд рекомендацій щодо вдосконалення програмного продукту:

- рольове розмежування доступу - доцільно реалізувати повноцінну модель ролей з правами доступу на основі груп або ролей користувачів (адміністратор, ревізор, перегляд тощо). Це дозволить обмежити функціональність згідно з посадовими обов’язками;
- інтеграція зі сторонніми сервісами - рекомендовано передбачити API для інтеграції з бухгалтерськими та ERP-системами (наприклад, SAP, BAS, M.E.Doc), що дозволить синхронізувати дані про активи, документи та фінансові операції;
- розширення системи сповіщень - запропоновано реалізувати відправку автоматичних повідомлень електронною поштою або в месенджери при зміні відповідального, статусу техніки, або наближенні дати інвентаризації;
- розширення можливостей аналітики - корисним стане впровадження модуля дашбордів, діаграм та звітів на основі даних про рух техніки, її кількість, стан, відповідальних осіб тощо;

- впровадження ведення журналу дій (audit trail) - доцільно створити окремий інтерфейс для перегляду історії змін із фільтрами за користувачем, об'єктом, датою та типом дії;
- можливість експорту/імпорту в інших форматах - планується розширення можливостей експорту в PDF та DOCX, а також підтримка шаблонних звітів і пакетного імпорту даних з інших систем;
- впровадження багатомовності - для потенційного використання в багатомовному середовищі передбачити механізм локалізації текстових елементів інтерфейсу;
- використання AI-аналітики та генерації звітів - з метою покращення управлінських рішень та аналітики, можна реалізувати модуль, що використовуватиме штучний інтелект (наприклад, GPT або ML-моделі) для аналізу великих обсягів даних, виявлення аномалій, автоматичного складання звітів про стан технічного парку, тенденції зміни відповідальності або навантаження на підрозділи. Це дозволить формувати прогнози та рекомендації для управління IT-інфраструктурою.

ВИСНОВКИ

Досліджено сучасні підходи до обліку та управління технічними ресурсами на підприємствах. Виявлено типові проблеми, серед яких: відсутність актуалізованих даних, дублювання інформації, складність інтеграції між підрозділами та нестача автоматизованих рішень.

Визначено технічні й організаційні бар'єри впровадження єдиної системи управління ресурсами. Сформульовано функціональні та нефункціональні вимоги до web-застосунку, які охоплюють продуктивність, масштабованість, захищеність даних, гнучкість у роботі з різними категоріями користувачів.

Обґрунтовано вибір технологій: фреймворку Laravel для реалізації серверної логіки, Vue.js для створення клієнтської частини, MySQL — як надійної СУБД з підтримкою складних зв'язків між сутностями. Вибраний стек дозволяє створити ефективну, розширювану та безпечну систему.

Спроєктовано й реалізовано структуру бази даних, яка охоплює сутності користувачів, активів, кабінетів, статусів, історії дій. Забезпечено логіку CRUD-операцій, автентифікацію через Active Directory, централізовану обробку даних, та інтеграцію з механізмами безпеки.

Проведено тестування застосунку, під час якого перевірено відповідність реалізованого функціоналу поставленим вимогам. Здійснено усунення виявлених помилок, оцінено стабільність і зручність роботи системи.

Оцінено потенціал застосунку до впровадження у корпоративному середовищі. Запропоновано напрями подальшого вдосконалення, зокрема розширення прав доступу, додавання звітності та інтеграції з внутрішніми ERP або helpdesk-системами.

ПЕРЕЛІК ПОСИЛАНЬ

1. Vite. *The Build Tool for the Web*. URL: https://vite.dev/.
2. Home - snipe-it open source IT asset management. *Home - Snipe-IT Free open source IT asset management*. URL: https://snipeitapp.com/.
3. Asset panda: the #1 asset management platform. *Asset Panda*. URL: https://www.assetpanda.com/.
4. InvGate asset management - IT asset management tool. *InvGate - Build a state-of-the-art IT operation*. URL: https://invgate.com/asset-management.
5. OpenMAINT. *Property & Facility Management*. URL: https://www.openmaint.org.
6. JetBrains. WebStorm: IDE JetBrains для JavaScript и TypeScript. *JetBrains*. URL: https://www.jetbrains.com/ru-ru/webstorm/.
7. The most-comprehensive AI-powered DevSecOps platform | GitLab. *GitLab*. URL: https://about.gitlab.com/.
8. PHP: hypertext preprocessor. *PHP: Hypertext Preprocessor*. URL: https://www.php.net/.
9. JavaScript | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript.
10. Laravel - the PHP framework for web artisans. *Laravel - The PHP Framework For Web Artisans*. URL: https://laravel.com/.
11. Active directory domain services overview. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview.
12. Using HTTP cookies - HTTP | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Cookies.
13. Vue.js. *Vue.js - The Progressive JavaScript Framework | Vue.js*. URL: https://vuejs.org/.

14. What is vuex? | vuex. *What is Vuex?* | *Vuex*. URL: https://vuex.vuejs.org/.
15. Починаючи | axios docs. *Axios*. URL: https://axios-http.com/uk/docs/intro.
16. MySQL. *MySQL*. URL: https://www.mysql.com/.
17. SPA (single-page application) - MDN web docs glossary: definitions of web-related terms | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA.
18. Json. *JSON*. URL: https://www.json.org/json-en.html.
19. Eloquent ORM - laravel 5.0 - the PHP framework for web artisans. *Laravel - The PHP Framework For Web Artisans*. URL: https://laravel.com/docs/5.0/eloquent.
20. Що таке DNS: просте пояснення принципів роботи та значення. *DNS*. URL: https://ipnet.ua/blog/chto-takoe-dns.
21. Inventory management systems: a comprehensive review and analysis / K. Pandey et al. *Interantional journal of scientific research in engineering and management*. 2023. Vol. 07, no. 11. P. 1–11. URL: https://doi.org/10.55041/ijsrem26979.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи:** Покращення процесів обліку, інвентаризації та управління технічними засобами компанії шляхом створення функціонального web-застосунку.

- **Об'єкт дослідження:** Процес управління матеріальними та технічними ресурсами підприємства.

- **Предмет дослідження**: Web-застосунок для інтеграції процесів інвентаризації та управління технічними засобами компанії.

2

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Дослідити особливості процесів обліку та управління технічними ресурсами в сучасних компаніях.
2. Визначити основні проблеми інтеграції інвентаризаційних процесів з управлінськими системами.
3. Сформувати перелік функціональних та нефункціональних вимог до web-застосунку.
4. Обґрунтувати вибір технологій для реалізації клієнтської та серверної частини системи.
5. Розробити базу даних для збереження інформації про ресурси компанії.
6. Виконати тестування на коректність функціонування застосунку.
7. Проаналізувати ефективність впровадженого рішення та надати рекомендації щодо його подальшого розвитку.

3

АНАЛІЗ АНАЛОГІВ

Показник	Snipe-IT	Asset Panda	InvGate Assets	OpenMAINT	EVR
Ліцензія	Open Source	Commercial	Commercial	Open Source	Commercial
Хмарне середовище	-	+	+	-	+
API / інтеграції	+	+	+	+	+
Кастомізація	-	-	-	-	+
Придатність до малих підприємств	+	-	-	-	+
Локалізація / підтримка мов	Англійська	Англійська	Англійська	Англійська	Англійсько-Українська
Простота впровадження	Середня	Висока	Низька	Низька	Висока

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

- Реєстрація та авторизація користувачів (Можливість реєструватися та входити в систему).
- Облік технічних засобів(Користувач може додавати нове обладнання, редагувати інформацію, прив'язувати обладнання до відповідального співробітника або кабінету).
- Інвентаризація (Можливість формувати списки для проведення інвентаризації, позначення фактичної наявності, стану, місцезнаходження).
- Управління статусами(Відображення стану обладнання: використовується, на складі, в ремонті, списано).
- Експорт та імпорт даних(Підтримка форматів Excel/CSV для обміну інформацією).

Нефункціональні вимоги:

- Захист персональних даних користувачів, шифрування паролів.
- Стабільна робота без збоїв і втрати даних.
- Можливість розширення функціоналу.
- Оптимізація запитів до бази даних для великого обсягу інформації.
- Відгук системи не більше ніж 2-3 секунди.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Фреймворк Laravel

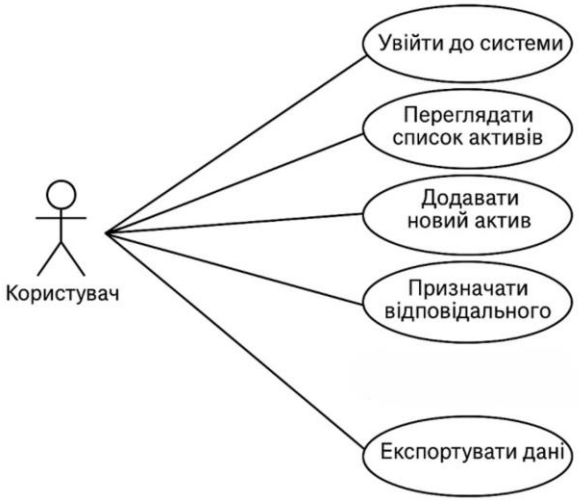
Builder Vite

Фреймворк
Vue.js

База даних
MySQL

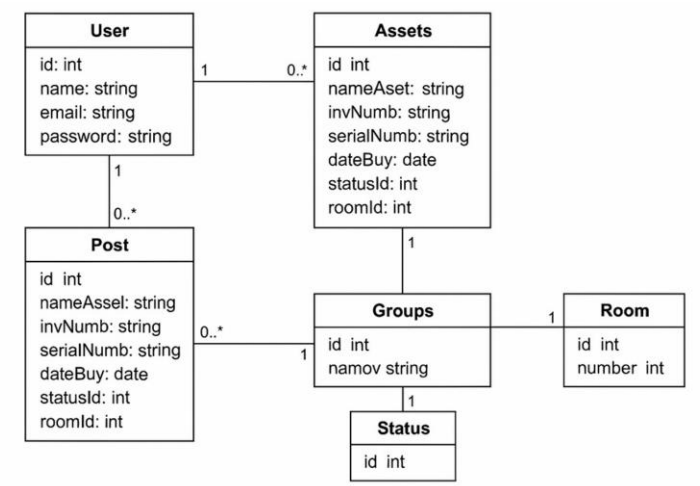
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



7

ДІАГРАМА КЛАСІВ

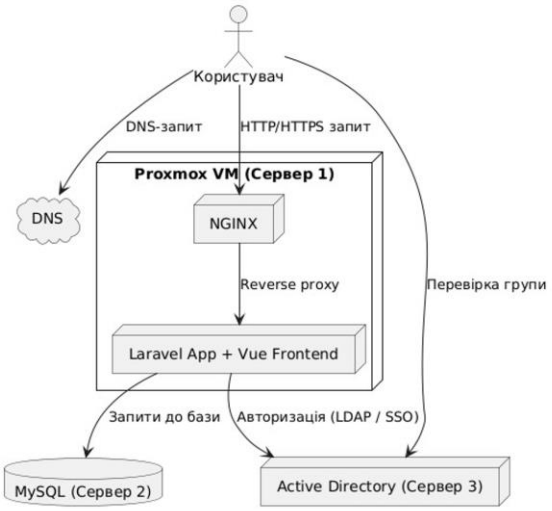


8

ДІАГРАМИ АКТИВНОСТЕЙ

9

ДІАГРАМИ ВПРОВАДЖЕННЯ

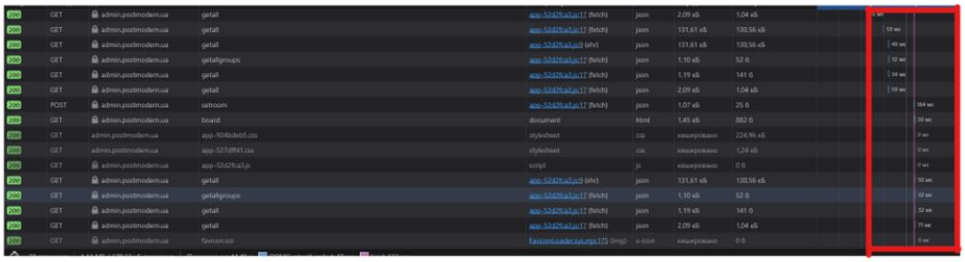


10

ТЕСТУВАННЯ

id	name	email	email_verified_at	password	remember_token
1	admin	admin@postmodern.ua	NULL	$2y$10$iKYmULREQtVr[redacted]A7IEvgxXJq	NULL
2	Dmytro Prilipko	d.prylipko@postmodern.ua	NULL	$2y$10$QCKEdF3XFdfH[redacted]4ud9gBNFTy	NULL

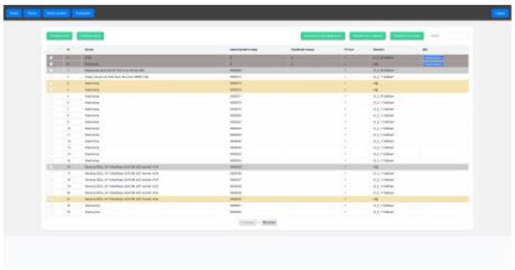
Шифровані паролі облікового запису



Час відгуку

11

ЕКРАННІ ФОРМИ



Головний екран додатку

Екран списку кімнат

Екран авторизації в додатку

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Приліпко Д.М., Андрусевич Н.В. ВИЗНАЧЕННЯ ВИМОГ ДО ВЕБЗАСТОСУНКА ДЛЯ ІНТЕГРАЦІЇ ПРОЦЕСІВ ІНВЕНТАРИЗАЦІЇ ТА УПРАВЛІННЯ ТЕХНІЧНИМИ ЗАСОБАМИ КОМПАНІЇ З ВИКОРИСТАННЯМ ФРЕЙМВОРКІВ VUE.JS ТА LARAVEL // Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, ДУІКТ, Київ, Україна, с.74-75.

2. Приліпко Д.М., Андрусевич Н.В. БЕЗПЕКА ВЕБЗАСТОСУНКУ ДЛЯ ІНТЕГРАЦІЇ ПРОЦЕСІВ ІНВЕНТАРИЗАЦІЇ ТА УПРАВЛІННЯ ТЕХНІЧНИМИ ЗАСОБАМИ КОМПАНІЇ З ВИКОРИСТАННЯМ ФРЕЙМВОРКІВ VUE.JS ТА LARAVEL. // Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025». 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с. 317-319.

13

ВИСНОВКИ

1. Проаналізовано сучасний стан обліку та управління технічними ресурсами — проведене дослідження дозволило виявити ключові особливості та проблеми, з якими стикаються компанії під час обліку та управління ресурсами.
2. Визначено основні труднощі інтеграції інвентаризаційних процесів з управлінськими системами — зосереджено увагу на технічних і організаційних бар'єрах, що ускладнюють впровадження єдиної системи управління.
3. Розроблено детальні вимоги до web-застосунку — сформульовано як функціональні (що має робити система), так і нефункціональні вимоги (продуктивність, безпека, зручність користування тощо), що забезпечують якість продукту.
4. Обґрунтовано вибір відповідних технологій — на основі аналізу сучасних інструментів визначено оптимальний стек технологій для реалізації клієнтської та серверної частин системи.
5. Створено базу даних для обліку ресурсів компанії — розроблено структуру зберігання даних, яка забезпечує зручний доступ і можливість масштабування в майбутньому.
6. Проведено тестування працездатності застосунку — перевірено коректність реалізації функціоналу, стабільність роботи системи та відповідність заявленим вимогам.
7. Оцінено ефективність впровадженого рішення — проаналізовано результати використання системи та сформульовано практичні рекомендації щодо її подальшого вдосконалення.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
<?php

namespace App\Http\Controllers;
use App\Models\Assets;
use App\Models\Groups;
use Illuminate\Http\Request;

class AssetController extends Controller
{
   //
   public function getAll()
   {
      $allAssets =
Assets::select('id','name','invNumb','serialNumb','st
atusId','roomId')->where('inGroup', '=', '0')->get()-
>toArray();
      $allGroups =
Groups::select('id','name','invNumb','serialNumb','s
tatusId','roomId')->get()->toArray();

      $toAnswer = array_merge( $allGroups,
$allAssets);

      return response()->json($toAnswer);
   }

   public function getDefault()
   {
      $tu = [
         [
            'id' => '1',
            'nameAsset' => 'Клавіатура',
            'invNumb' => '000956324',
            'dateBuy' => '01-02-2023',
            'statusId' => '1',
            'roomId' => '1',
         ],
         [
            'id' => '2',
            'nameAsset' => 'Мишка',
            'invNumb' => '00324366',
            'dateBuy' => '01-02-2022',
            'statusId' => '1',
            'roomId' => '2',
         ],
      ];

      return response()->json($tu);
   }
}
```

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller
{
   /**
    * Create a new controller instance.
    *
    * @return void
    */
   public function __construct()
   {
      $this->middleware('auth');
   }

   /**
    * Show the application dashboard.
    *
    * @return
\Illuminate\Contracts\Support\Renderable
    */
   public function index()
   {
      return view('home');
   }
}
```

```
<?php

namespace App\Models;

use
Illuminate\Database\Eloquent\Factories\HasFactor
y;
use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
   use HasFactory;

   protected $fillable = [
      'nameAsset', 'invNumb' ,'title', 'body',
   ];
}
```

```
import axios from 'axios';

const state = {
  page: 1, // Текущая страница
  perPage: 20,
  total: 0,
  items: [],
  loading: false,
  searchQuery: '',
};

const getters = {
  displayedItems: state => {
    const start = (state.page - 1) * state.perPage;
    const end = start + state.perPage;
    const filteredItems = state.items.filter(item =>
      Object.values(item).some(value =>

String(value).toLowerCase().includes(state.search
Query.toLowerCase())
      )
    );
    return filteredItems.slice(start, end);
  },
};

const actions = {
  fetchItems({ commit }) {
    axios.get('http://localhost:8000/assets/getall')
      .then(response => {
```

```
        commit('setItems', response.data);
      })
      .catch(error => {
        console.log(error);
      });
  },
  setPage({ commit }, page) {
    commit('setPage', page);
  },
};

const mutations = {
  setItems(state, items) {
    state.items = items;
    state.total = items.length;
  },
  setPage(state, page) {
    state.page = page;
  },
  setSearchQuery(state, query) {
    state.searchQuery = query;
  },
};

export default {
  namespaced: true,
  state,
  getters,
  actions,
  mutations,
};

import './bootstrap';

import {createApp} from 'vue';
import App from './App.vue';
import {store} from '../store/Index';
import Index from './components/Index.vue';
import Board from './components/Board.vue';
import router from './router';

createApp(App).use(store).use(router).component('
index', Index).component('board',
Board).mount("#app")

import { createRouter, createWebHistory } from
'vue-router';
import Login from './Components/Login.vue';
import Registration from
'./Components/Registration.vue';
import Board from './Components/Board.vue';

const routes = [
    {
      path: '/board',
      name: 'get.board',
      component: Board,
    },
    {
        path: '/user/login',
        name: 'user.login',
        component: Login,
    },
    {
        path: '/user/registration',
        name: 'user.registration',
        component: Registration,
    },
];

const router = createRouter({
  history: createWebHistory(),
  routes,
});

export default router;

<template #header>
  <div>
    <router-link :to="{ name:
'user.login'}">Login</router-link>
    <router-link :to="{ name:
'user.registration'}">Registration</router-link>
    <router-link :to="{ name:
'get.board'}">Board</router-link>

    <router-view></router-view>
  </div>
</template>

<script>

</script>

<script setup>
import { computed, onMounted, ref } from 'vue';
import { useStore } from 'vuex';

const currentPage = computed(() =>
store.state.assets.page);
const totalPages = computed(() =>
Math.ceil(store.state.assets.total /
store.state.assets.perPage));

const pageNumber = ref('');

const changePage = (page) => {
  store.dispatch('assets/setPage', page);
};

const goToPage = () => {
  const page = parseInt(pageNumber.value);
  if (!isNaN(page) && page > 0 && page <=
totalPages.value) {
    changePage(page);
    pageNumber.value = '';
  }
};

const store = useStore();

onMounted(() => {
  store.dispatch('assets/fetchItems');
});

const displayedItems = computed(() =>
store.getters['assets/displayedItems']);

const updateSearchQuery = (event) => {
  store.commit('assets/setSearchQuery',
event.target.value);
  store.dispatch('assets/fetchItems');
};

const assets = computed(() =>
store.getters['assets/allItems']);

</script>
```

```
<template>
<div class="py-12">
  <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
     <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg">
        <div class="p-6 bg-white border-b border-gray-200">
           <div class="flex items-center justify-between mb-6">
              <div>
                <!-- :href="route('posts.create')" -->
                <a class="px-6 py-2 text-white bg-green-500 rounded-md focus:outline-none">
                  Create Post
                </a>
              </div>
              <div>
                <form class="flex">
                  <input
                    class="px-4 py-2 border border-gray-400 rounded-md focus:outline-none"
                    type="text"
                    placeholder="Search"
                    v-model="searchQuery"
                    @input="updateSearchQuery"
                  >
                </form>
              </div>
            </div>


          <div style="height: 680px;">
          <table className="table-fixed w-full">
             <thead>
                <tr className="bg-gray-100">
                   <th className="px-4 py-2 w-20">No.</th>
                   <th className="px-4 py-2">Name</th>
                   <th className="px-4 py-2">Inventory Number</th>
                   <th className="px-4 py-2">Serial Number</th>
                   <th className="px-4 py-2">Status</th>
                   <th className="px-4 py-2">Room</th>
                   <th className="px-4 py-2">Action</th>
                </tr>
             </thead>
             <tbody>
                <tr v-for="(asset, index) in displayedItems" :key="index">
                   <td class="border px-4">{{ asset.id }}</td>
                   <td class="border px-4">{{ asset.name }}</td>
                   <td class="border px-4">{{ asset.invNumb }}</td>
                   <td class="border px-4">{{ asset.serialNumb }}</td>
                   <td class="border px-4">{{ asset.statusId }}</td>
                   <td class="border px-4">{{ asset.roomId }}</td>
                   <td class="border px-4">
                      <!-- Добавьте ваши компоненты для каждой ячейки -->
                      <button tabIndex="1" className="px-4 py-1 text-sm text-white bg-blue-500 rounded" :href="123">
                        Edit
                      </button>
                      <button @click="123" tabIndex="-1" type="button" className="mx-1 px-4 py-1 text-sm text-white bg-red-500 rounded">
                        Delete
                      </button>
                    </td>
                    </tr>
                </tbody>
            </table>
            </div>
            <div>
  </div>

  <div class="flex items-center justify-center mt-5">
    <button
      v-for="page in totalPages"
      :key="page"
      class="px-4 py-1 ml-2 text-sm text-white bg-gray-400 rounded"
      :class="{ 'bg-blue-500': page === currentPage }"
      @click="changePage(page)"
    >
      {{ page }}
    </button>
  </div>
        </div>
     </div>
  </div>
</div>
</template>

<script setup>
import { computed, onMounted, onUnmounted, ref } from 'vue';

const props = defineProps({
   align: {
      type: String,
      default: 'right',
   },
   width: {
      type: String,
      default: '48',
   },
   contentClasses: {
      type: String,
      default: 'py-1 bg-white',
   },
});

const closeOnEscape = (e) => {
   if (open.value && e.key === 'Escape') {
      open.value = false;
   }
};

onMounted(() =>
document.addEventListener('keydown', closeOnEscape));
onUnmounted(() =>
```

```
document.removeEventListener('keydown',
closeOnEscape));

const widthClass = computed(() => {
  return {
    48: 'w-48',
  }[props.width.toString()];
});

const alignmentClasses = computed(() => {
  if (props.align === 'left') {
    return 'origin-top-left left-0';
  } else if (props.align === 'right') {
    return 'origin-top-right right-0';
  } else {
    return 'origin-top';
  }
});

const open = ref(false);
</script>

<template>
  <div class="relative">
    <div @click="open = !open">
      <slot name="trigger" />
    </div>

    <!-- Full Screen Dropdown Overlay -->
    <div v-show="open" class="fixed inset-0 z-
40" @click="open = false"></div>

    <transition
      enter-active-class="transition ease-out
duration-200"
      enter-from-class="transform opacity-0
scale-95"
      enter-to-class="transform opacity-100
scale-100"
      leave-active-class="transition ease-in
duration-75"
      leave-from-class="transform opacity-100
scale-100"
      leave-to-class="transform opacity-0 scale-
95"
    >
      <div
        v-show="open"
        class="absolute z-50 mt-2 rounded-md
shadow-lg"
        :class="[widthClass, alignmentClasses]"
        style="display: none"
        @click="open = false"
      >
        <div class="rounded-md ring-1 ring-
black ring-opacity-5" :class="contentClasses">
          <slot name="content" />
        </div>
      </div>
    </transition>
  </div>
</template>
```