НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

«До захисту допущено»

Завідувач кафедри

__________ Євгенія СУЛЕМА

«___» ____________ 2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Програмне забезпечення для фіксування майна орендаря та орендодавця під час надання послуг з оренди»

Виконав:
студент IV курсу, групи КП-11
Кучеренко Дмитро Олегович ___________

Керівник:
асистент кафедри ПЗКС, д-р філософії,
Юсин Яків Олексійович ___________

Консультант з нормоконтролю:
доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович ___________

Рецензент:
доцент кафедри ПСТ ФІТ «КНУ
ім. Тараса Шевченка», д.т.н., с.н.с.,
Порєв Геннадій Володимирович ___________

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____________

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

________ Євгенія СУЛЕМА

«___»_____________2024 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Кучеренку Дмитру Олеговичу

1. Тема проєкту «Програмне забезпечення для фіксування майна орендаря та орендодавця під час надання послуг з оренди», керівник проєкту Юсин Яків Олексійович, асистент кафедри ПЗКС, доктор філософії, затверджені наказом по університету №1808-С від «29» травня 2025 р.

2. Термін подання студентом проєкту «13» червня 2025 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   – огляд та аналіз наявних рішень;
   – обґрунтування вибору засобів реалізації;
   – розроблення програмного забезпечення;
   – аналіз розробленого програмного забезпечення.

5. Перелік обов'язкового графічного матеріалу:
   – структура бази даних (креслення);
   – алгоритм адресації зовнішніх запитів (креслення);
   – схема архітектури кожного мікросервісу (плакат);
   – схема етапів логування (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2025 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	15.11.2024	
2.	Розроблення та узгодження технічного завдання	26.11.2024	
3.	Розроблення структури вебзастосунку	10.12.2024	
4.	Підготовка першого розділу дипломного проєкту	28.12.2024	
5.	Розроблення дизайну сторінок та графічних елементів	05.02.2025	
6.	Підготовка другого розділу дипломного проєкту	22.02.2025	
7.	Програмна реалізація вебзастосунку	15.03.2025	
8.	Тестування вебзастосунку	17.03.2025	
9.	Підготовка третього розділу дипломного проєкту	29.03.2025	
10.	Підготовка четвертого розділу дипломного проєкту	18.04.2025	
11.	Підготовка графічної частини дипломного проєкту	24.04.2025	
12.	Оформлення документації дипломного проєкту	28.05.2025	

Студент Дмитро КУЧЕРЕНКО

Керівник проєкту Яків ЮСИН

АНОТАЦІЯ

Даний дипломний проєкт присвячено розробленню програмного забезпечення для фіксування початкового стану об’єкта оренди, відслідковування зміни в процесі використання та документування кінцевого стану при поверненні майна. Основною метою проєкту є покращення якості документування стану майна під час процесу орендування.

Шляхом комплексного дослідження існуючих рішень у сфері обліку майна було виявлено їхні суттєві обмеження та сформовано унікальну концепцію продукту. Ключовим підходом стала реалізація системи дошок – цифрових каталогів приміщень з інтерактивними можливостями опису предметів та відстеження їхнього статусу. Особливу увагу приділено функціональності для співпраці користувачів, що дозволяють залучати до роботи з дошками всіх зацікавлених сторін, зокрема власників приміщень до орендарів.

При виконанні дипломного проєкту було розроблено програмне забезпечення з чітким розподілом функціональності між спеціалізованими сервісами. Рішення включає ретельно спроєктовані бази даних для цілісного збереження інформації, серверну частину з оптимізованою бізнес-логікою та інтуїтивно зрозумілий користувацький інтерфейс. Впроваджено надійний механізм аутентифікації та багаторівневий контроль доступу до даних для різних груп користувачів. Окрім основної функціональності для роботи з майном, система надає можливості переглядати історію змін, налаштовувати надсилання сповіщень та отримувати їх. Також користувачі можуть персоналізовувати свій профіль та змінювати облікові дані.

ABSTRACT

This diploma project is dedicated to the development of software for recording the initial state of rental properties, tracking changes during use, and documenting the final condition upon return of the property. The main goal of the project is to improve the quality of documentation regarding property conditions throughout the rental process.

Through comprehensive research of existing property management solutions, significant limitations were identified, leading to the formation of a unique product concept. The key approach was the implementation of a board system – digital catalogs of premises with interactive capabilities for describing items and tracking their status. Special attention was paid to user collaboration functionality, allowing all stakeholders, particularly property owners and tenants, to be involved in working with the boards.

In the process of completing the diploma project, software was developed with a clear distribution of functionality between specialized services. The solution includes carefully designed databases for comprehensive information storage, a server component with optimized business logic and an intuitive user interface. A reliable authentication mechanism and multi-level access control for different user groups were implemented. In addition to the core functionality for property management, the system provides capabilities to view change history, configure notification settings, and receive alerts. Also users can personalize their profiles and modify their account information.

ДП.045440-01-90 Програмне забезпечення для фіксування майна орендаря та орендодавця під час надання послуг з оренди. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Програмне забезпечення	5	
	для фіксування майна		
	орендаря та орендодавця		
	під час надання послуг з		
	з оренди. Технічне		
	завдання		
ДП.045440-03-81	Програмне забезпечення	61	
	для фіксування майна		
	орендаря та орендодавця		
	під час надання послуг з		
	з оренди. Пояснювальна		
	записка		
ДП.045440-04-51	Програмне забезпечення	4	
	для фіксування майна		
	орендаря та орендодавця		
	під час надання послуг з		
	з оренди. Програма та		
	методика тестування		
ДП.045440-05-34	Програмне забезпечення	12	
	для фіксування майна		
	орендаря та орендодавця		
	під час надання послуг з		
	з оренди. Керівництво		
	користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Програмне забезпечення	1	
	для фіксування майна		
	орендаря та орендодавця		
	під час надання послуг з		
	з оренди. Схема бази		
	даних. ER-діаграма		
ДП.045440-07-99	Програмне забезпечення	1	
	для фіксування майна		
	орендаря та орендодавця		
	під час надання послуг з		
	з оренди. Алгоритм		
	адресації зовнішніх		
	запитів. Блок-схема		
	алгоритму		
ДП.045440-08-98	Програмне забезпечення	1	
	для фіксування майна		
	орендаря та орендодавця		
	під час надання послуг з		
	з оренди. Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

__________ Євгенія СУЛЕМА

“___” ____________ 2024 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ФІКСУВАННЯ МАЙНА ОРЕНДАРЯ ТА ОРЕНДОДАВЦЯ ПІД ЧАС НАДАННЯ ПОСЛУГ З ОРЕНДИ

Технічне завдання

ДП.045440-02-91

“ПОГОДЖЕНО”

Керівник проєкту:

__________ Яків ЮСИН

Нормоконтроль:

__________ Микола ОНАЙ

Виконавець:

________ Дмитро КУЧЕРЕНКО

2024

ЗМІСТ

1. Найменування та галузь застосування.. 3

2. Підстава для розроблення..3

3. Призначення розробки... 3

4. Вимоги до програмного продукту..3

5. Вимоги до проєктної документації...4

6. Етапи проєктування..4

7. Порядок тестування розробки...5

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: програмне забезпечення для фіксування майна орендаря та орендодавця під час надання послуг з оренди.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для створення зручного інструменту для покращення якості документування стану майна під час процесу орендування.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Програмне забезпечення повинно забезпечувати такі основні функції:

1. Створення облікового запису.
2. Вхід до облікового запису.
3. Оновлення персональних даних.
4. Створення нової дошки.
5. Редагування майна дошки.
6. Перегляд історії змін дошки.
7. Редагування користувачів дошки.
8. Редагування опису майна дошки.

9. Вихід з дошки або її видалення.
10. Перегляд, прочитання та видалення сповіщень.
11. Налаштування отримання сповіщень.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   – «Структура бази даних. UML-діаграма».
   – «Алгоритм адресації зовнішніх запитів. Блок-схема».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою проєкту………………………13.11.2023

Розроблення та узгодження технічного завдання…………………25.11.2023

Розроблення структури вебзастосунку…………………………….15.12.2023

Підготовка матеріалів розділу дипломного проєкту………………30.12.2024

Розроблення дизайну сторінок та графічних елементів…………..03.02.2025

Підготовка другого розділу дипломного проєкту…………….……19.02.2025

Програмна реалізація вебзастосунку………………………………10.03.2024

Тестування вебзастосунку……………………………………………..17.03.2025

Підготовка третього розділу дипломного проєкту………………...30.03.2025

Підготовка четвертого розділу дипломного проєкту………..……..12.04.2025

Підготовка графічної частини дипломного проєкту………..……...21.04.2025

Оформлення документації дипломного проєкту…………………..26.05.2025

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

__________ Євгенія СУЛЕМА

"___" ____________ 2025 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ФІКСУВАННЯ МАЙНА ОРЕНДАРЯ ТА ОРЕНДОДАВЦЯ ПІД ЧАС НАДАННЯ ПОСЛУГ З ОРЕНДИ

Пояснювальна записка

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проєкту:

__________ Яків ЮСИН

Нормоконтроль:

__________ Микола ОНАЙ

Виконавець:

_______ Дмитро КУЧЕРЕНКО

2025

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ.................................3
ВСТУП...6
1. ОГЛЯД ТА АНАЛІЗ НАЯВНИХ РІШЕНЬ... 7
1.1. Аналіз особливостей документування майна під час оренди............. 7
1.2. Традиційні паперові акти...8
1.3. Електронні нотатники зі спільним доступом...................................... 9
1.4. Застосунок MRI Property Tree Connect.. 10
1.5. Застосунок Rental Property Management..12
1.6. Висновки.. 14
2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ............................ 15
2.1. Технології для розроблення серверної частини.................................15
2.2. Системи керування базами даних.. 17
2.3. Технології для розроблення клієнтської частини.............................19
2.4. Інфраструктура автоматизації та розгортання..................................21
2.5. Висновки.. 24
3. РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ............................25
3.1. Загальний опис застосунку..25
3.2. Аналіз функціональних вимог до програмного забезпечення.......... 26
3.3. Особливості реалізації серверної частини застосунку...................... 28
3.4. Структура бази даних...35
3.5. Особливості реалізації клієнтської частини застосунку...................45
3.5. Висновки.. 47
4. АНАЛІЗ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ............49
4.1. Аналіз реалізованого застосунку... 49
4.2. Тестування реалізованої серверної частини застосунку....................50
4.3. Тестування реалізованої клієнтської частини застосунку.................52
4.4. Рекомендації щодо подальшого вдосконалення................................ 54
4.5. Висновки.. 55
ВИСНОВКИ...57
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ.......................... 59
ДОДАТКИ... 61

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Конкурентність (Concurrency) – властивість систем, коли декілька процесів обчислення відбуваються водночас, і, можливо, взаємодіють один з одним.

Горутини (Goroutine) – це потік виконання, яким керує середовище виконання Go.

Компіляція – трансляція початкового коду програми в бінарний код.

Перехоплювач (Interceptor) – це проміжне програмне забезпечення, яке перехоплює та змінює запити та відповіді API, перш ніж вони досягають місця призначення.

API (Application programming interface) – це спосіб взаємодії комп'ютерних програм між собою.

Абстракції – це набір структур даних та функцій вищого рівня, що спрощують роботи з функціональністю нижчого рівня.

Генератори коду – це інструменти, які здатні створювати файли з кодом певної мови базуючись на певній схемі, яка задає правила генерації.

Буферизація – метод зберігання інформації на обчислювальній машині, іноді тимчасово.

СКБД (Система керування базами даних) – набір взаємопов'язаних даних і програм для доступу до цих даних. Надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних.

Реляційна база даних – це сукупність елементів даних, організованих у вигляді набору формально описаних таблиць, з яких дані можуть бути доступними або повторно зібрані багатьма різними способами без необхідності реорганізації таблиць бази даних.

Транзакція баз даних – група послідовних операцій з базою даних, що можуть бути виконані цілком і успішно, дотримуючись цілісності даних

і незалежно від інших транзакцій, або не виконані зовсім, і тоді вона не справить ніякого ефекту.

ACID – це набір властивостей, що гарантують надійну роботу транзакцій бази даних: атомарність, узгодженість, ізольованість, довговічність.

Фреймворк – інфраструктура програмних рішень, що полегшує розробку складних систем.

CI/CD пайплайни – це автоматизована послідовність дій, яка дозволяє інтегрувати, тестувати та розгортати оновлення програмного забезпечення з максимальною ефективністю.

Авторизатор (Authorizer) – це спосіб контролю доступу до API. Коли клієнт робить запит до методу API, API Gateway викликає авторизатор.

Обмежувач частоти запитів (Rate limiter) – це обмеження швидкості, що використовується для керування швидкістю запитів, які надсилаються чи приймаються контролером мережевого інтерфейсу.

VPC (Virtual Private Cloud) – це віртуальна приватна хмара, що надає повний контроль над віртуальним мережевим середовищем, включаючи розподіл ресурсів, підключення та безпеку..

Graceful Shutdown – це коли комп’ютер вимикається програмною функцією, що дозволяє операційній системі виконати свої завдання з безпечного завершення процесів та закриття з’єднань.

Перевірка справності API (Health check) – це метод моніторингу API, діагностичний інструмент, який перевіряє API та сповіщає, коли помічає проблеми.

SPA (Single-page application) – це реалізація вебзастосунку, яка завантажує лише один вебдокумент, а потім оновлює вміст тіла цього єдиного документа за допомогою JavaScript API тоді, коли потрібно відобразити інший вміст.

SSR (Server-side rendering) – вся сторінка відображається на сервері при першому запиті, а потім уже в браузері ініціалізуються клієнтські компоненти і далі застосунок працює як звичайний SPA.

Black Box – це метод тестування програмного забезпечення, який перевіряє функціональність програми без заглиблення в її внутрішню структуру чи роботу та не має до неї доступ.

SSE (Server-sent events) – це технологія надсилання сервером, яка дозволяє клієнту отримувати автоматичні оновлення від сервера після встановлення початкового клієнтського з’єднання.

ВСТУП

У сучасному суспільстві ринок оренди житла та комерційних приміщень стрімко розвивається, що призводить до збільшення кількості орендних відносин між власниками майна та орендарями. Проте традиційні методи документування стану майна при передачі в оренду часто бувають неповними, неструктурованими або взагалі не використовуються, що створює передумови конфліктних ситуацій.

Незважаючи на наявність певних інструментів для фіксації стану орендованого майна, існує нагальна потреба в комплексному рішенні, яке б усувало недоліки наявних підходів та забезпечувало прозорість і захист інтересів усіх сторін орендних відносин.

Метою цього дипломного проєкту є покращення якості документування стану майна під час процесу орендування. Для досягнення цієї мети пропонується створити інтуїтивно зрозумілий інструмент «Smartner», що дозволить детально фіксувати початковий стан об'єкта оренди, відслідковувати будь-які зміни в процесі використання та документувати кінцевий стан при поверненні майна.

Використовуючи зручний інтерфейс та розширені функціональні можливості, користувачі зможуть створювати детальні інвентаризаційні описи з фотоматеріалами, фіксувати пошкодження в реальному часі, отримувати підтвердження від обох сторін щодо стану майна та зберігати історію змін.

Застосунок «Smartner» допоможе запобігати конфліктам між орендодавцями та орендарями та підвищить рівень відповідальності користувачів орендованого майна та створить прозорі умови для всіх учасників процесу. Завдяки чіткій документації стану майна користувачі зможуть уникати безпідставних звинувачень, зберігати кошти та час на вирішення суперечок, а також формувати позитивну репутацію як відповідальні орендарі чи орендодавці.

1. ОГЛЯД ТА АНАЛІЗ НАЯВНИХ РІШЕНЬ

1.1. Аналіз особливостей документування майна під час оренди

Документування стану майна є невід’ємною частиною орендних відносин. Воно відіграє ключову роль для захисту інтересів обох сторін, дозволяючи орендодавцям та орендарям уникати непорозумінь, фінансових втрат та юридичних конфліктів. Традиційні паперові акти прийому-передачі та фотофіксація використовуються в орендних відносинах протягом багатьох років, однак розвиток цифрових технологій і сучасні потреби все більше сприяють цифровізації цього процесу.

Останнім часом стрімко зростає потреба у впровадженні спеціалізованих застосунків для документування стану майна. Це зумовлено низкою факторів.

Ринок оренди в сучасних умовах зазнає значних трансформацій, характеризуючись підвищенням мобільності орендних відносин. Динамічність ринку вимагає швидкого та зручного способу фіксації стану майна без необхідності використання паперових документів. Особливої актуальності це набуває через зростання популярності короткострокової оренди та використання цифрових платформ для пошуку житла [1].

Технологічний прогрес відіграє ключову роль у модернізації орендних процесів, адже сьогодні переважна більшість орендодавців та орендарів мають доступ до смартфонів. Ці пристрої стають ефективним інструментом для детальної фіксації стану майна, дозволяючи створювати фотодокументацію в єдиній системі замість використання розрізнених паперових актів [1], що часто губляться або пошкоджуються з часом.

Паралельно з технологічними змінами зростають і вимоги до прозорості орендних відносин. Учасники ринку потребують надійних інструментів, які забезпечують можливість створення, зберігання та постійного доступу до документації про стан майна. Важливим аспектом таких рішень є можливість підтвердження достовірності інформації обома

сторонами, що мінімізує ризик виникнення конфліктних ситуацій.

Цифрові рішення також відкривають нові можливості для структурованого зберігання даних. Систематизація інформації про стан майна за різними категоріями, приміщеннями та типами пошкоджень суттєво спрощує процес порівняння стану об’єкта до і після оренди. Це не лише оптимізує робочі процеси для орендодавців, але й забезпечує орендарям більше впевненості та прозорості щодо умов оренди.

1.2. Традиційні паперові акти

Паперові акти прийому-передачі та інвентаризаційні описи протягом багатьох років є основним інструментом в орендних відносинах для фіксації стану майна. Орендодавці та орендарі використовують їх для опису стану стін, підлоги, меблів, побутової техніки тощо.

Традиційні паперові акти приймання-передачі житла пропонують орендодавцям та орендарям базові переваги для регулювання їхніх відносин. Документування стану майна на початку та в кінці оренди дозволяє мати точку відліку для порівняння. Ключовою перевагою таких актів є їхня юридична значущість – фізичний документ з підписами обох сторін має юридичну силу при виникненні спорів і може бути представлений у суді як офіційний доказ.

Однак паперові акти мають суттєві недоліки, що обмежують їх ефективність у сучасних умовах. Вони характеризуються недостатньою детальністю опису майна та відсутністю якісного візуального підтвердження, що ускладнює визначення часу виникнення пошкоджень. Паперові документи можуть бути втрачені, пошкоджені чи з часом стати менш читабельними через деформацію матеріалу, що ставить під загрозу доступ до важливої інформації. Також наявні такі обмеження, як складність внесення змін при виявленні нових пошкоджень під час оренди та необхідність одночасної фізичної присутності обох сторін для підписання, що не завжди можливо в сучасному динамічному світі.

1.3. Електронні нотатники зі спільним доступом

Електронні спільні нотатники, такі як Google Docs, Microsoft OneNote, Samsung Note, є певним кроком уперед у порівнянні з паперовими актами документування стану майна при оренді [2]. Орендодавці та орендарі використовують їх для створення текстових описів, завантаження фотографій та спільного доступу до інформації про стан майна.

Використання електронних нотатників для документування стану орендованого майна має значні переваги порівняно з традиційними паперовими методами. Електронні системи забезпечують зручність спільного доступу, дозволяючи обом сторонам орендних відносин переглядати та редагувати документи без необхідності організовувати фізичні зустрічі чи обмінюватися фізичними копіями. Це суттєво спрощує процес комунікації та взаємодії між орендодавцем та орендарем. Важливою перевагою також є можливість прикріпляти практично необмежену кількість якісних фотоматеріалів, що значно покращує візуальну документацію стану майна та дозволяє зафіксувати найдрібніші деталі, які могли б залишитися непоміченими в паперовому форматі.

Однак використання електронних нотатників має й суттєві недоліки, що обмежують їхню ефективність протягом оренди. Відсутність стандартизованості призводить до неструктурованого документування, де різні елементи майна можуть бути описані з різним рівнем деталізації або бути взагалі пропущеними, що ускладнює систематичний аналіз стану приміщення. Загальні електронні рішення зазвичай пропонують обмежені можливості відслідковування історії змін, не забезпечуючи детальної інформації про конкретні модифікації документа, час їх внесення та авторство. Відсутність спеціалізованих функцій для порівняння початкового та поточного стану майна і категоризації пошкоджень ускладнює ефективне використання таких систем для орендних цілей. Додатково виникають проблеми з довгостроковим зберіганням даних – доступ до документів може бути втрачений через необережні маніпуляції,

такі як зміна параметрів доступу або випадкове видалення, оскільки загальні електронні нотатники не призначені спеціально для ведення довгострокових облікових записів.

1.4. Застосунок MRI Property Tree Connect

MRI Property Tree Connect являє собою спеціалізований мобільний застосунок, спрямований на управління нерухомістю та документування її стану [3]. Цей застосунок пропонує розширені можливості для орендодавців, але має певну специфіку використання, яка визначає його переваги та недоліки.

Переваги застосунку управління нерухомістю полягають у його структурованому підході до моніторингу стану приміщень, що забезпечує систематичне планування та відстеження технічного обслуговування об’єктів. Цей застосунок пропонує зручні інструменти для організації та контролю регулярних прибирань та інших послуг з обслуговування приміщень. Він також включає комплексні функції для відстеження фінансових операцій, пов’язаних з орендою, автоматичні нагадування про майбутні платежі та заплановані заходи з обслуговування (рис. 1.1). Додатковою цінністю є можливість впровадження з бухгалтерськими системами, що суттєво полегшує процеси фінансового обліку та звітності для власників значної кількості нерухомості.

Однак використання цього застосунку має й певні недоліки, такі як обмеженість функціональності щодо детального документування стану рухомого майна в приміщенні. Він не забезпечує можливості створення візуальної бази даних пошкоджень та змін стану окремих предметів, а також не містять спеціалізованих інструментів для порівняння початкового та поточного стану інвентаря. Процес отримання облікового запису достатньо складний, а висока вартість ліцензії робить такі системи економічно невиправданими для дрібних орендодавців. Системи управління нерухомістю фокусуються переважно на процесах управління

будівлями в цілому, а не на захисті інтересів конкретних сторін у контексті збереження стану майна. Також відсутні спрощені інструментів комунікації між орендодавцем та орендарем та відсутній вебінтерфейс, що суттєво погіршує зручність використання системи з різних пристроїв і операційних систем, що не підтримують встановлення даного програмного забезпечення.

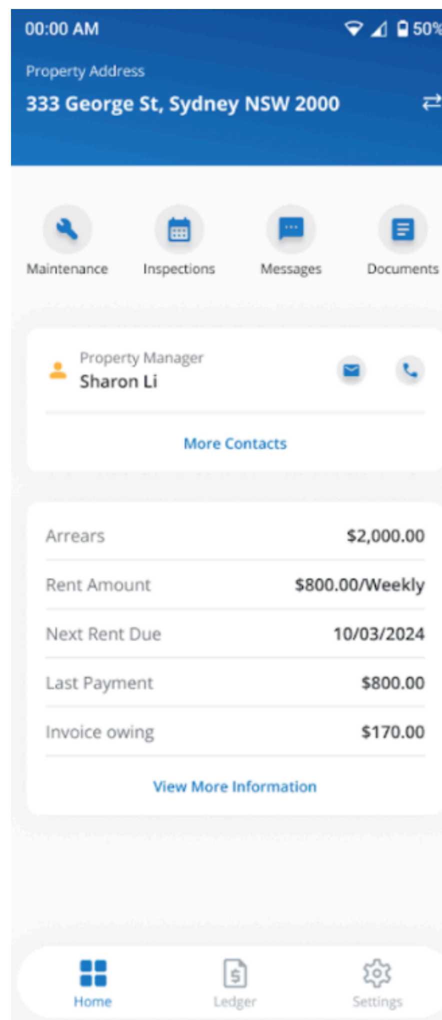


Рис. 1.1. Інтерфейс приміщення застосунку MRI Property Tree Connect

MRI Property Tree Connect в першу чергу розроблений для великих гравців ринку нерухомості, таких як агентства з управління нерухомістю та готельні мережі. Для таких користувачів його функціональність є доречною, проте для приватних орендодавців та орендарів, які потребують простого та ефективного інструменту документування стану майна, цей

застосунок виявляється надмірно складним, дорогим та не є спрямованим на вирішення їхніх потреб.

1.5. Застосунок Rental Property Management

Rental Property Management представляє собою програмне рішення, призначене для автоматизації процесів управління орендною нерухомістю [4]. Цей застосунок має власний підхід до організації роботи з орендними об’єктами, що визначає його сильні та слабкі сторони.

Переваги застосунку для управління орендою полягають у всебічній функціональності для адміністрування орендних відносин. Він пропонує інтегровану систему обліку орендних платежів з функцією автоматичного нагадування про заборгованості, що дозволяє ефективно контролювати фінансові аспекти. Централізоване зберігання контактної інформації орендарів та історії комунікацій з ними забезпечує зручний доступ до важливих даних. Користувачі також отримують можливість створювати шаблони типових документів для оформлення орендних відносин, ця особливість суттєво економить час та стандартизує процеси. Додатково наявна функціональність для відстеження статусу вакантних та орендованих приміщень у режимі реального часу, систему формування фінансової звітності за різними періодами та категоріями витрат, а також інструменти для управління запитами на технічне обслуговування від орендарів.

Однак використання цього застосунку має й певні недоліки. Надмірна зосередженість на адміністративних та фінансових аспектах оренди, а не на документуванні фактичного стану майна, ускладнює користувацький досвід малих орендодавців. Відсутність спеціалізованих функцій для створення детальних описів та фотофіксації стану рухомого майна та його порівняння на різних етапах оренди також не відповідає необхідним потребам. Система надає обмежені можливості для категоризації та пошуку по елементам інвентаря та їхнього стану, що

унеможливлює відстеження змін. Процедура початкового налаштування виявляється складною і потребує специфічних знань у сфері та значних часових витрат. Застосунок характеризується примітивним та застарілим користувацьким інтерфейсом (рис. 1.2), що не відповідає сучасним стандартам UX/UI, особливо для осіб з обмеженими фізичними можливостями. Це випливає у незручності користування системою навігації, що вимагає численних кліків для доступу до базових функцій.

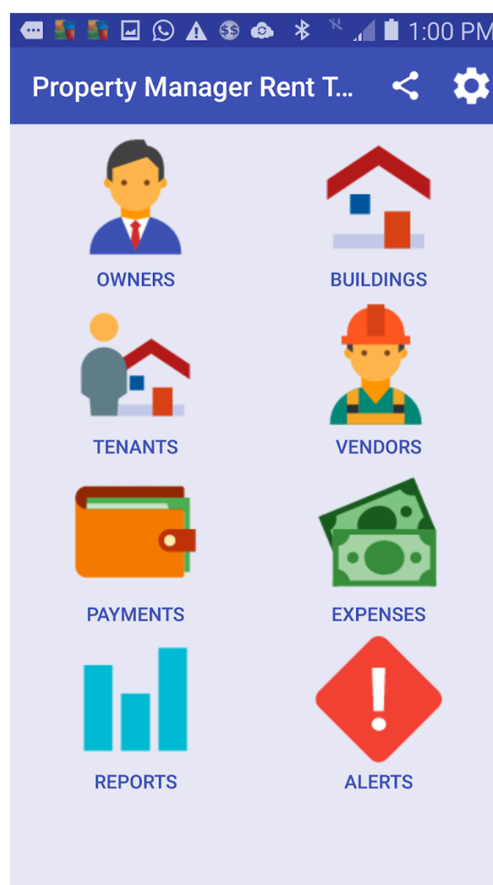


Рис. 1.2. Інтерфейс меню застосунку Rental Property Management

Rental Property Management орієнтований переважно на професійних менеджерів нерухомості, агентства та компанії, що керують множинними орендними об’єктами. Для таких користувачів його функціональність у сфері адміністрування та фінансового контролю є корисною, проте застарілий дизайн та неінтуїтивний інтерфейс створюють значні перешкоди для ефективної роботи. Для звичайних орендодавців та

орендарів цей застосунок виявляється надто складним у використанні та не вирішує проблему забезпечення прозорого і зручного документування стану майна для запобігання потенційним конфліктам.

1.6. Висновки

Проведений аналіз наявних способів документування стану майна при оренді демонструє, що ця сфера потребує інноваційних підходів, оскільки традиційні методи та існуючі цифрові рішення не забезпечують повноцінного вирішення проблеми. Будь-який новий застосунок, що прагне зайняти свою нішу в цьому сегменті, повинен враховувати виявлені недоліки та запропонувати дійсно ефективне рішення для всіх учасників орендних відносин.

На даний момент ринок пропонує декілька варіантів документування стану майна – від традиційних паперових актів, які мають обмеження щодо рівня детальності опису та часто не містять достатнього візуального підтвердження, до електронних нотатників із проблемами стандартизації та відстеження змін. Спеціалізовані програми, такі як MRI Property Tree Connect та Rental Property Management, зосереджуються переважно на фінансово-адміністративних аспектах та не відповідають потребам документування стану рухомого майна.

Базуючись на проведеному аналізі, було прийнято рішення розробити вебзастосунок, який подолає обмеження наявних систем документування майна при оренді та запропонує орендодавцям і орендарям ефективний інструмент для прозорої фіксації стану об’єктів оренди протягом всього орендного циклу. Для досягнення успіху новий застосунок має запропонувати баланс між простотою використання та комплексністю функціональність.

2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Технології для розроблення серверної частини

Вибір технологій для серверної частини – стратегічне рішення, що впливає на весь життєвий цикл проєкту. Правильно обрана технологія забезпечує не лише виконання функціональних вимог, але й оптимізує швидкість розробки, масштабованість та операційні витрати. Помилки на цьому етапі часто призводять до технічного боргу або повного переписування системи. Технологія має також відповідати запланованій інфраструктурі. Обґрунтований вибір, що базується на аналізі вимог та альтернатив, створює надійну основу для конкурентоспроможної програмної системи.

2.1.1. Go та мікросервісна архітектура

Go або Golang [5] – сучасна мова програмування з відкритим кодом, розроблена компанією Google. Завдяки своїм особливостям, Go стала популярним вибором для реалізації мікросервісної архітектури та використовується багатьма компаніями, включаючи Docker, Kubernetes, Uber та Dropbox.

Go вирізняється високою продуктивністю, оскільки компілюється безпосередньо в машинний код, що забезпечує швидке виконання програм. Вбудована підтримка конкурентності за допомогою горутин дозволяє ефективно обробляти численні запити одночасно, що є критичним для мікросервісної архітектури. Мінімалістичний синтаксис полегшує написання та підтримку коду, а статична типізація допомагає знаходити помилки ще на етапі компіляції.

Важливою перевагою Go є її лаконічність та ідіоматичність. Синтаксис мови спроєктовано таким чином, щоб максимально спростити написання чистого, зрозумілого коду, що легко підтримувати та масштабувати. Мова заохочує використання композиції замість

успадкування, що сприяє розробці гнучких логічних одиниць з чітко визначеними інтерфейсами та мінімальною зв'язаністю.

Була обрана мікросервісна архітектура [6], адже вона забезпечує гнучкість, масштабованість та стійкість системи, оскільки відмова одного сервісу не призводить до падіння всієї системи. Кожен мікросервіс відповідає за конкретну бізнес-функцію та може бути розроблений, розгорнутий і масштабований незалежно від інших компонентів. Це дозволяє швидше випускати оновлення та ефективніше використовувати обчислювальні ресурси. В межах кожного мікросервісу шарова (layered) архітектура має багато переваг, адже вона забезпечує чітке розділення відповідальності та спрощує розробку й тестування. Верхній шар API обробляє вхідні запити та формує відповіді, сервісний шар містить бізнес-логіку та координує роботу між різними компонентами, а шар доступу до даних інкапсулює логіку роботи з базами даних та взаємодію з іншими сервісами.

Програми, що написані на Go, мають невеликий розмір, що дозволяє запускати більше сервісів на тій самій інфраструктурі. Це особливо важливо в контексті мікросервісів, де може працювати значна кількість невеликих компонентів одночасно. Швидка компіляція Go також прискорює цикл розробки та розгортання, що є великою перевагою при частих оновленнях, характерних для такої архітектури.

2.1.2. Засоби комунікації

Зручність зовнішньої комунікації клієнту з сервером забезпечується за допомогою легкості RESTful API та впроваджується з бібліотекою Gin, яка забезпечує високу продуктивність завдяки оптимізованому маршрутизатору та мінімальному споживанню пам'яті. Бібліотека пропонує вбудовані механізми для валідування запитів, оброблення помилок, логування та реалізації перехоплювачів, що значно спрощує розробку надійних API. Абстракції Gin дозволяють описувати маршрути та

обробники лаконічно та зрозуміло, а функціональність групування маршрутів сприяє структурованій організації маршрутизаторів.

Для забезпечення безпечної внутрішньої комунікації між мікросервісами використовується gRPC, адже сама технологія була розроблена Google з урахуванням особливостей цієї мови. Використання Protocol Buffers для визначення контрактів сервісів забезпечує сувору типізацію та ефективне кодування даних, що критично для високонавантажених систем. Генератори коду автоматично створюють клієнтські та серверні інтерфейси на основі proto-файлів, мінімізуючи рутинну роботу. Вбудована підтримка горутин ідеально поєднується з потоковими можливостями gRPC, дозволяючи створювати високопродуктивні двонаправлені потоки даних.

Для кращого розділення відповідальності мікросервісів та забезпечення асинхронної комунікації був обраний брокер повідомлень Kafka, який зручно впроваджується завдяки наявності зрілих клієнтських бібліотек. Горутини Go забезпечують ефективну обробку повідомлень у фоновому режимі, не блокуючи основний потік виконання. Можливість не безпосередньої, а розподіленої комунікації, дозволяє сервісам еволюціонувати незалежно з мінімальним впливом змін у одному сервісі на інші. Крім того, брокер забезпечує буферизацію повідомлень, що підвищує стійкість системи до пікових навантажень та тимчасової недоступності окремих компонентів.

Комбінація цих технологій комунікації [7] в мікросервісах забезпечує гнучку, масштабовану архітектуру, де кожен механізм взаємодії використовується відповідно до своїх сильних сторін.

2.2. Системи керування базами даних

Вибір системи керування базами даних є критичним аспектом архітектури, що безпосередньо впливає на продуктивність та надійність програмного рішення. Оптимальні СКБД мають забезпечувати цілісність

даних навіть при високих навантаженнях, ефективне логування для відслідковування стану та помилок, належну швидкість обробки запитів. При виборі необхідно зважати на специфіку даних проєкту, моделі їх використання та вимоги до масштабування. Компроміс між реляційними та нереляційними підходами повинен базуватися на балансі між потребою в структурованості та гнучкості схеми даних, враховуючи поточні та прогнозовані обсяги інформації.

2.2.1. Реляційна СКБД PostgreSQL

PostgreSQL – це гнучка об’єктно-реляційна система керування базами даних [8] з відкритим кодом, що має понад 30 років активного розвитку. Ця СКБД працює на всіх основних операційних системах, включаючи Linux, Windows та macOS, забезпечуючи високу кросплатформність, та підтримує розгортання як на локальних серверах, так і в контейнерах та хмарних середовищах.

Реляційні бази даних є оптимальним вибором для систем, де необхідно гарантувати цілісність та узгодженість даних. Модель відношень забезпечує структуровану організацію інформації з чітко визначеними зв’язками між сутностями, що запобігає дублюванню даних. Транзакція гарантує атомарність операцій, захищаючи базу від часткового оновлення при збоях. Основними перевагами PostgreSQL є висока відповідність стандартам SQL, здатність обробляти складні робочі навантаження, підтримка розширень та повнотекстового пошуку. Система забезпечує надійні механізми для підтримки цілісності даних через первинні та зовнішні ключі, унікальні обмеження, тригери та транзакції, що відповідають принципам ACID. Це критично важливо для бізнес-застосунків, де цілісність інформації є пріоритетом.

PostgreSQL має широку екосистему інструментів для керування, моніторингу та міграції даних, а також активну спільноту розробників. Для невеликих проєктів важливою перевагою є відсутність ліцензійних

обмежень, що дозволяє розпочати розробку без витрат на ліцензування з можливістю подальшого масштабування за потреби.

2.2.2. Індексова СКБД Elasticsearch та візуалізація аналітики

Elasticsearch – розподілений пошуковий та аналітичний рушій на базі Apache Lucene. Система підтримує кластеризацію з горизонтальним масштабуванням, працює на різних платформах та легко розгортається в хмарних середовищах.

Ключовими перевагами Elasticsearch для логування є швидкість обробки великих обсягів даних у реальному часі та потужний повнотекстовий пошук. Система оптимізована для напівструктурованих даних, що робить її ідеальною для централізованого логування в мікросервісній архітектурі. У поєднанні з Kibana технічні дані перетворюються на інформативні візуалізації та дошки. Інструмент дозволяє створювати діаграми, часові графіки та налаштовувати моніторинг у реальному часі. Особливо цінними є можливості сповіщень та виявлення аномалій.

Разом Elasticsearch та Kibana формують ефективний інструментарій для відстеження продуктивності, швидкого виявлення та розслідування проблем у складних розподілених системах.

2.3. Технології для розроблення клієнтської частини

Технології для клієнтської частини визначають якість взаємодії користувача з системою та суттєво впливають на рівень залученості та задоволеності сервісом. Оптимальне рішення має забезпечувати адаптивність інтерфейсу до різних пристроїв та розмірів екранів, високу реактивність навіть при нестабільному з’єднанні, та інтуїтивно зрозумілий користувацький досвід. Вибір фреймворку повинен враховувати такі фактори як продуктивність на цільових пристроях, швидкість початкового

завантаження, а також доступність інструментів для тестування та моніторингу користувацького інтерфейсу.

2.3.1. Фреймворк Next.js

Next.js – це React-фреймворк з відкритим кодом, розроблений Vercel, що забезпечує потужну інфраструктуру для створення сучасних вебзастосунків. Фреймворк підтримує різні методи рендерингу, включаючи серверний рендеринг (SSR), статичну генерацію (SSG) та клієнтський рендеринг, що дозволяє оптимізувати продуктивність та SEO, в той час як інкрементальна статична регенерація (ISR) дає можливість оновлювати статичний контент без необхідності повної перебудови сайту, що особливо корисно для динамічних даних, які змінюються нечасто. Вбудована маршрутизація на основі файлової системи спрощує навігацію у застосунку без необхідності підключення додаткових бібліотек.

Бібліотека React Query значно спрощує організацію API клієнтів у Next.js застосунках. Ця бібліотека автоматизує складні аспекти управління даними, такі як кешування, повторні запити та фонове оновлення, та ефективно відокремлює серверний стан від клієнтського, пропонуючи інтуїтивні хуки для обробки різних статусів запитів, включаючи завантаження, помилки та оптимістичні оновлення. Такий підхід суттєво зменшує кількість коду для впровадження з RESTful API та підвищує надійність застосунку.

Для додаткового керування реактивним станом Zustand пропонує елегантне децентралізоване рішення. На відміну від більш складних альтернатив, як-от Redux, ця бібліотека має мінімалістичну реалізацію без надмірного шаблонного коду та природно інтегрується з хуками React. Zustand забезпечує реактивні оновлення компонентів при зміні стану та підтримує гнучкі селектори для оптимізації рендерингу в складних інтерфейсах.

Поєднання Next.js з зазначеними технологіями створює гармонійну екосистему для розробки застосунків з чітким розділенням відповідальності та забезпечує прогресивний користувацький досвід.

2.3.2. Стилізації компонентів дизайну з Tailwind

Tailwind CSS революціонізує підхід до стилізації вебзастосунків завдяки своєму утилітарному підходу. Замість написання власних CSS-правил, розробники використовують готові класи безпосередньо в HTML, що значно прискорює процес розробки. Цей підхід зменшує когнітивне навантаження, дозволяючи працювати в одному файлі замість постійного перемикання між HTML та CSS.

Система адаптивного дизайну в Tailwind елегантно вирішує проблему різних розмірів екранів, а підтримка темної теми робить впровадження кольорових схем інтуїтивно зрозумілим. Завдяки оптимізованому розміру вихідних файлів, бібліотека забезпечує високу продуктивність без компромісів у гнучкості стилізації.

Для складніших інтерактивних елементів інтерфейсу чудовим доповненням є бібліотека shadcn/ui. Вона пропонує колекцію доступних компонентів, стилізованих з Tailwind. Замість імпорту готових рішень, shadcn/ui дозволяє скопіювати компоненти безпосередньо в проєкт, забезпечуючи повний контроль над їхньою модифікацією. Компоненти бібліотеки вирішують складні задачі інтерактивності з мінімальними зусиллями та відповідають сучасним стандартам доступності.

Поєднання Tailwind для базової стилізації та shadcn/ui для інтерактивних елементів створює збалансований інструментарій, що прискорює розробку без обмеження творчої свободи дизайну.

2.4. Інфраструктура автоматизації та розгортання

Інфраструктурні рішення формують фундамент для розгортання та підтримки програмних систем, визначаючи їхню стабільність та

економічну ефективність. Правильно підібрана інфраструктура забезпечує гнучкість у побудові мікросервісної архітектури, оптимізуючи незалежне розгортання компонентів та горизонтальне масштабування. Важливими факторами є прозора цінова політика з можливістю передбачення витрат при зростанні навантаження, підтримка децентралізованої архітектури для підвищення стійкості системи, а також наявність інструментів для автоматизації розгортання, моніторингу та реагування на інциденти.

2.4.1. Сервіс для спільної розробки та автоматизації Github

Ефективна система контролю версій є фундаментальним елементом сучасного процесу розробки програмного забезпечення. Вона дозволяє інтегровувати різні гілки коду, забезпечуючи відстеження змін, можливість відкату до попередніх станів та захист від випадкової втрати роботи. Правильно налаштована система контролю версій підвищує прозорість розробки та спрощує виявлення та виправлення помилок.

GitHub пропонує багатий набір інструментів для управління кодом. Розділення на гілки дозволяє групувати код відповідно до логічного значення, у той час як функціональність пул-реквестів забезпечує структурований підхід до перегляду коду, стимулюючи огляд рішення та підвищуючи якість програмного забезпечення.

Особливо цінним аспектом є впровадження GitHub Actions – системою автоматизації робочих процесів. Ці пайплайни дозволяють автоматизувати критичні етапи розробки, такі як тестування, збірка та розгортання. Налаштувавши ці дії, можна гарантувати, що кожна зміна в репозиторії проходить стандартизовані процедури перевірки якості. Автоматизовані процеси суттєво підвищують продуктивність, звільняючи розробників від рутинних завдань та зменшуючи кількість людських помилок. Вони забезпечують стабільний процес впровадження та розгортання програмного забезпечення, та конфігурованість робить процес збірки та розгортання документованим, відтворюваним та незалежним.

Впровадження належної системи контролю версій та автоматизацію робочих процесів є стратегічним кроком, що значно підвищує якість програмного забезпечення, пришвидшує цикл розробки та створює більш здорове та продуктивне середовище.

2.4.2. Інфраструктура розгортання AWS

Amazon Web Services пропонує комплексну інфраструктуру [9] для розгортання та управління мікросервісною архітектурою, забезпечуючи оптимальний баланс між надійністю, безпекою та масштабованістю. Фундаментом захищеного середовища служить Amazon Virtual Private Cloud (VPC), що дозволяє створити ізольовану мережу з повним контролем над IP-адресацією та маршрутизацією та налаштування вхідний та вихідний трафік ресурсів за допомогою групи безпеки.

Контейнеризація через Elastic Container Service (ECS) забезпечує стабільне, ізольоване середовище для кожного мікросервісу. Цей підхід гарантує однакову роботу застосунків незалежно від інфраструктури та спрощує процес оновлення без простоїв. Кожен контейнер містить усі необхідні залежності, що забезпечує чітке розмежування ресурсів та запобігає конфліктам між мікросервісами.

Гнучкість масштабування досягається завдяки комбінації Elastic Load Balancer (ELB) та Fargate [10]. Fargate дозволяє запускати контейнери без необхідності створення та управління базовими серверами, забезпечуючи оптимальну модель для stateless-архітектури, де кожен запит може бути оброблений будь-яким з горизонтально масштабованих екземплярів сервісу, коли ELB рівномірно розподіляє вхідний трафік між ними, автоматично виявляє несправні процеси та перенаправляє запити лише на здорові.

Принцип інфраструктури як коду реалізується через CloudFormation, що дозволяє декларувати всі ресурси хмарного середовища у вигляді шаблонів. Цей підхід забезпечує узгодженість конфігурацій між різними

середовищами, коли розгортання інтегровується з автоматизованими системами контролю версій, як CI/CD пайплайни. Такий підхід мінімізує ризики людських помилок при налаштуванні середовища та забезпечує стабільність при масштабуванні.

Впровадження цих технологій створює середовище, де мікросервіси можуть безпечно комунікувати між собою, при цьому залишаючись ізольованими та незалежно масштабованими залежно від зміни навантаження.

2.5. Висновки

В результаті аналізу вимог для розробки системи обрано стек технологій, що забезпечує оптимальний баланс продуктивності, масштабованості та швидкості розробки. Головним критерієм при виборі є їхня здатність вирішувати поставлені завдання.

Для створення серверної частини вирішено використовувати Go завдяки його високій продуктивності, вбудованій підтримці паралелізму, що спрощує реалізацію мікросервісної архітектури, яка забезпечує чітке розділення відповідальності, спрощує тестування, та дозволяє інтегрувати різні способи комунікації. Для роботи з даними обрано PostgreSQL як надійну реляційну базу та Elasticsearch для логування, що відповідає вимогам до цілісності та аналізу.

Основою для створення користувацького інтерфейсу обрано Next.js, що забезпечує зручне керування станом, в той час як за стилізацію відповідає Tailwind забезпечуючи адаптивність і простоту розробки.

Надійне та безпечне розгортання ресурсів забезпечується з AWS з використання контейнеризації сервісів, масштабування та декларативного опису інфраструктури шаблонами, що забезпечує контроль версій та автоматизацію розгортання.

3. РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Загальний опис застосунку

При першому запуску вебзастосунку користувач потрапляє на сторінку входу, де можна ввести свої облікові дані. Якщо система розпізнає пристрій як такий, з якого раніше виконувався вхід, користувач автоматично переспрямовується на головну сторінку. Для нових користувачів доступна опція створення облікового запису на сторінці реєстрації, де потрібно заповнити форму з основною інформацією.

На головній сторінці застосунку, що є першою опцією меню, знаходиться список дошок приміщень, до яких користувач має доступ. Дошка – інтерактивна сторінка, до якої може мати доступ група користувачів, та редагувати її відповідно до власної ролі. Список можна фільтрувати за допомогою пошуку за текстом та сортувати за часом останнього оновлення. Користувач може створити нову дошку приміщення, обрати існуючу або перейти до сторінки запрошень, які були отримані від інших користувачів.

При переході до конкретної дошки користувач потрапляє на сторінку зі списком майна, що до неї прикріплений, який також можна фільтрувати та сортувати. При обранні опції додавання майна користувач може заповнити форму з назвою, описом та іншою корисної інформацією. Зі списку також можливо перейти до сторінок загальної інформації дошки, редагування списку доданих користувачів з обраними ролями та перегляду активних запрошень. Важливою опцією є можливість перейти на сторінку історії змін дошки, яка надає інформацію про абсолютно всі взаємодії з даними.

Друга опція меню дозволяє переглядати останні персональні сповіщення. Таким чином користувач завжди знатиме про нові запрошення, зміни в дошках та оновлення системи. Сповіщення можна позначати як прочитані та видалити.

Остання опція меню відкриває сторінку налаштувань, чи користувач можна переглянути обрану інформацію. При обранні профілю користувач може переглянути персональні дані, оновити їх або змінити пароль, попередньо підтвердивши поточний. Сторінка налаштування сповіщень дозволяє контролювати отримання сповіщень за категоріями, проте певні з них є обов’язковими.

Для роботи застосунку використовується архітектура на основі розподілу відповідальності між серверною та клієнтською частинами. Серверна складова забезпечує бізнес-логіку, аутентифікацію користувачів, зберігання даних та захист інформації, у той час як клієнтська частина відповідає за реалізацію зручного інтерфейсу для взаємодії користувача із системою та здійснює необхідну комунікацію з сервером для обміну даними.

3.2. Аналіз функціональних вимог до програмного забезпечення

Було сформовано перелік функціональних вимог на основі аналізу існуючих рішень, котрі необхідні для повноцінного функціонування застосунку. Створення списку функціональних вимог посприяло визначенню пріоритетних задач та формуванню плану розроблення застосунку. Перелік функціональних вимог наведено в таблиці 3.1.

Таблиця 3.1

Функціональні вимоги до програмного забезпечення

Код вимоги	Назва вимоги	Опис вимоги
F-1	Створення облікового запису	При створенні облікового запису відвідувач повинен надати таку інформацію: ім’я та прізвище, унікальне користувацьке ім’я, унікальну пошту та складний пароль. Система перевіряє дані на відповідність встановленим критеріям.

Продовження табл. 3.1

F-2	Вхід до облікового запису	Користувач повинен надати інформацію: пошта та пароль. Система перевіряє відповідність введених даних згідно до зареєстрованих облікових записів.
F-3	Оновлення персональних даних	Користувач повинен мати можливість редагувати інформацію: прізвище, ім’я, пошту, пароль та зображення-аватар. Система перевіряє введені дані на відповідність встановленим критеріям.
F-4	Створення нової дошки	Користувач повинен мати можливість створити дошку приміщення з інформацією: назва, опис, загальне зображення. Система перевіряє введені дані на відповідність встановленим критеріям та створює дошку із зазначенням її власника.
F-5	Редагування майна дошки	Користувач повинен мати можливість додавати, редагувати назву, опис, власність та час отримання, прибирання майна, видаляти його.
F-6	Перегляд історії змін дошки.	Користувач повинен мати можливість переглядати всі внесені загальні зміни дошки, зміни майна, користувачів та керування учасниками, що мають записуватися в історію
F-7	Редагування користувачів дошки	Користувач повинен мати можливість запрошувати за допомогою пошуку за користувацьким іменем, видаляти, налаштовувати ролі учасників дошки відповідно до своїх дозволів.

Продовження табл. 3.1

F-8	Редагування опису дошки	Користувач повинен мати можливість редагувати назву та опис дошки, змінювати її зображення.
F-9	Вихід з дошки або її видалення	Користувач повинен мати можливість вийти з дошки, якщо він був до неї запрошений, або видалити дошку, якщо він є її власником.
F-10	Перегляд, прочитання та видалення сповіщень	Користувач повинен мати можливість переглядати персональні сповіщення, що базуються на певних подіях, позначати їх як прочитані та видаляти зі списку.
F-11	Налаштування отримання сповіщень	Користувач повинен мати можливість налаштовувати, які з груп сповіщень він хотів би отримувати, а які приховати.

3.3. Особливості реалізації серверної частини застосунку

Реалізація серверної складової застосунку відіграє визначальну роль у забезпеченні ефективності всієї системи, створюючи міцний фундамент для її надійного функціонування. Архітектурні рішення на рівні серверної логіки безпосередньо впливають на швидкодію обробки запитів та оптимальне використання обчислювальних ресурсів. Особлива увага повинна приділятися впровадженню багаторівневих механізмів захисту даних, що включають шифрування критичної інформації, валідацію вхідних даних та контроль прав доступу для запобігання несанкціонованим втручанням. Завдяки продуманій організації сервісів серверу застосунок набуває необхідної гнучкості для адаптації до мінливих потреб користувачів без суттєвої перебудови базової архітектури.

3.3.1. Загальна архітектура застосунку

На сьогоднішній день архітектурні рішення для побудови сучасних вебзастосунків поділяються на різні підходи: від класичного монолітного застосунку до модульних рішень проміжного типу. Після детального аналізу вимог до масштабованості, гнучкості та стійкості системи, перевагу було надано мікросервісній архітектурі, яка дозволяє створити екосистему незалежних, спеціалізованих компонентів з чітко визначеними межами відповідальності.

Мікросервісна архітектура забезпечує низку переваг, що стали вирішальними при її виборі для проєкту: можливість незалежного розгортання та оновлення окремих компонентів без переривання роботи всієї системи, краща ізоляція відмов, можливість різноманітного технологічного стеку для різних сервісів залежно від їхніх особливостей, а також простіше горизонтальне масштабування конкретних компонентів системи при зростанні навантаження.

Відповідно до функціональних вимог та доменної моделі системи, було виділено чотири ключові мікросервіси (рис. 3.1). «Users Service» відповідає за керування профілями користувачів та їхніми персональними даними. «Auth Service» забезпечує процеси аутентифікації та авторизації, керування сесіями та редагування паролів. «Boards Service» реалізує всю бізнес-логіку, пов’язану з управлінням дошками приміщень, включаючи їх створення, додавання майна, керування доступом учасників та отриманням історії. «Notifications Service» забезпечує централізовану обробку та зберігання різноманітних сповіщень.

Важливим аспектом архітектури стала реалізація різних підходів до комунікації. Для роботи із зовнішніми клієнтами сервіси відкривають HTTP маршрути, які дотримуються семантики RESTful API, що надає гнучку доступність. Для синхронної міжсервісної взаємодії gRPC є кращим рішенням, адже робота з буферами забезпечує типізований та полегшений обмін даними, хоч і потребує більшого початкового налаштування. Для

асинхронних операцій впроваджено відокремлений брокер з чергами, які дозволяють сервісам публікувати події та підписуватися на них без прямої залежності від статусу інших компонентів системи. Така гібридна модель комунікації підвищує стійкість системи до тимчасової недоступності окремих сервісів та забезпечує ефективну обробку потоків даних різної інтенсивності.

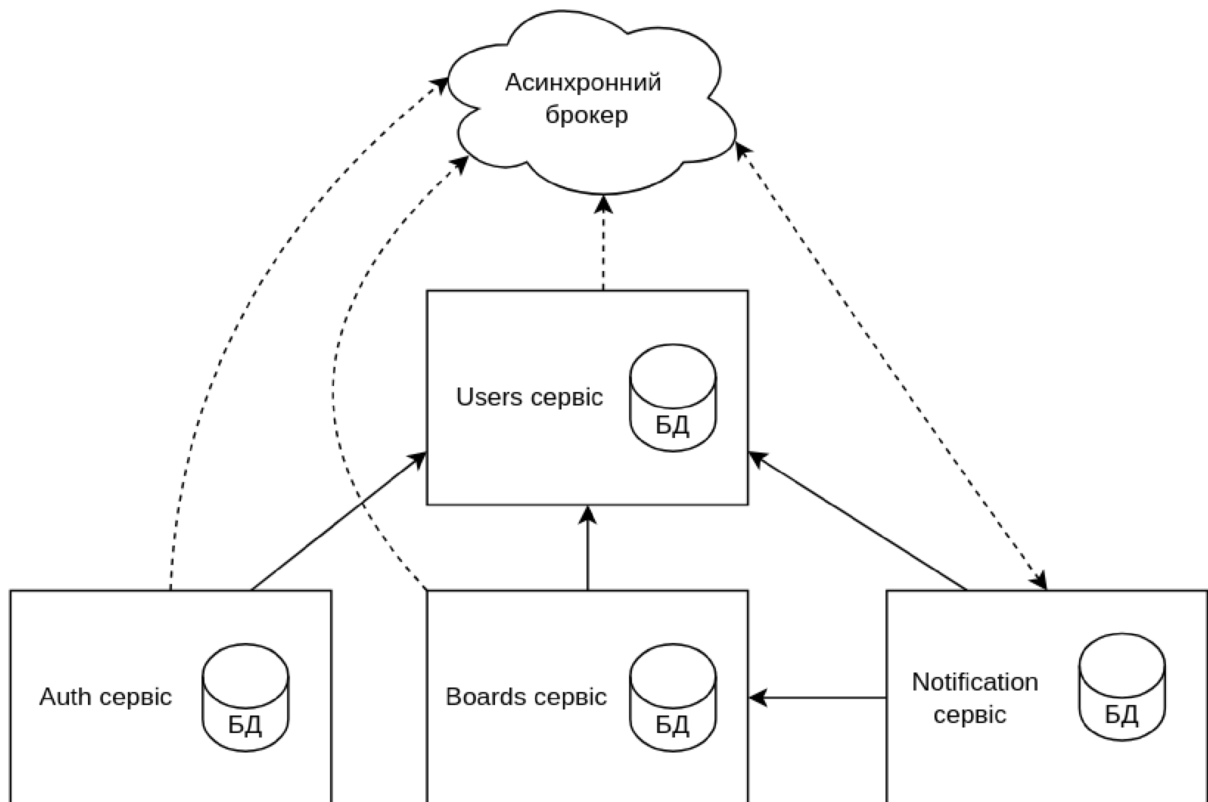


Рис. 3.1. Схема архітектури серверної частини

Кожен сервіс має власну базу даних, оптимально підібрану для характеру даних, що обробляються, та сценаріїв доступу до них. Така ізоляція даних забезпечує незалежність сервісів та полегшує підтримку цілісності в межах доменних меж. При необхідності розширення функціональності система може бути доповнена новими спеціалізованими мікросервісами без суттєвої перебудови існуючих компонентів.

3.3.2. Архітектура кожного мікросервісу застосунку

Для досягнення високої якості програмного забезпечення та гнучкості майбутнього розвитку системи було впроваджено принципи шарової архітектури, яка дозволяє чітко розмежувати відповідальність різних компонентів та забезпечити низьку зв'язаність між ними.

Архітектурний підхід спирається на різні концепти та адаптує їх до специфічних потреб проєкту. Згідно до цього підходу, мікросервіс можна поділити на чотири ключові шари, які комунікують за сталим принципом (рис. 3.2).

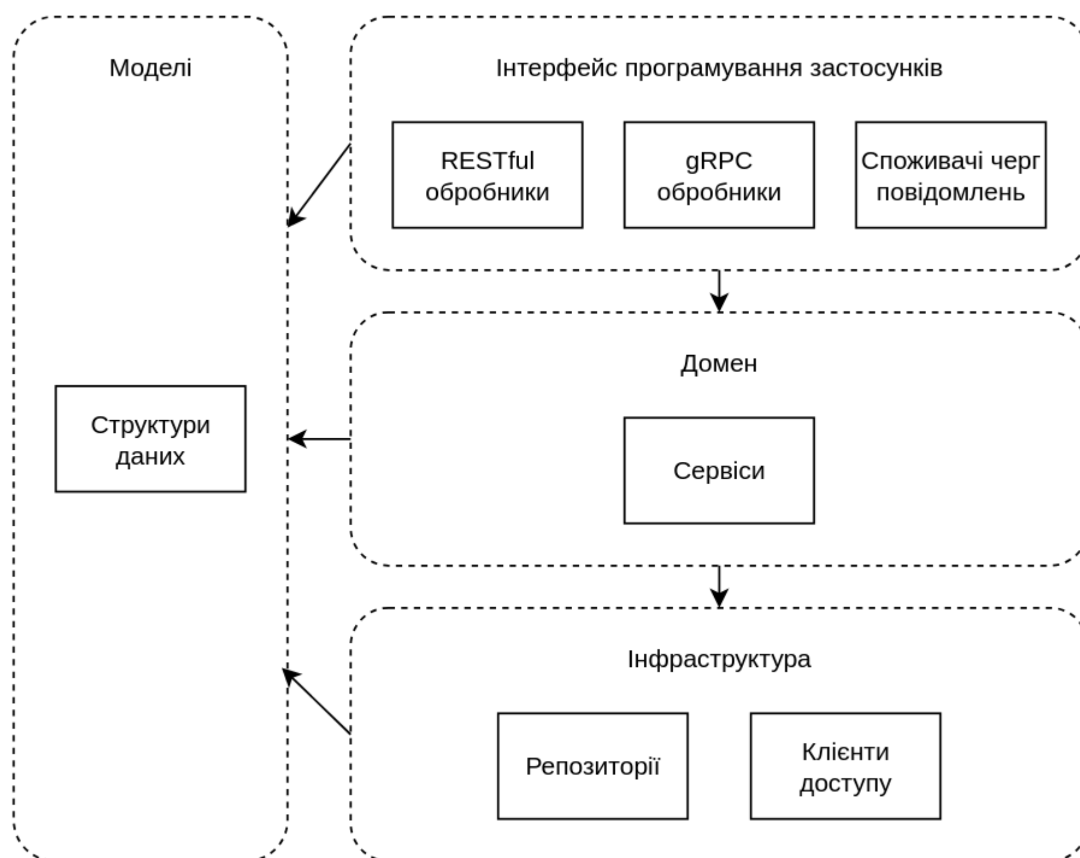


Рис. 3.2. Схема архітектури кожного мікросервісу

У фундаменті архітектурного рішення лежить шар «Models», який містить базові сутності та структури даних, інтерфейси компонентів, що використовуються всіма іншими шарами системи. Тут визначаються ключові об'єкти доменної області, які залежать від задачі сервісу: дошки про приміщення, їхнє майно, користувацькі профілі, сповіщення тощо. Ці

моделі є спільними для всієї системи та забезпечують уніфікований словник термінів, що використовується при комунікації між різними компонентами програмного забезпечення. Даний шар не містить жодної логіки, а лише описує структуру інформації.

Далі розташовується шар «Infrastructure», який відповідає за взаємодію з зовнішніми системами та збереження даних. У цьому шарі реалізовано репозиторії для роботи з різними джерелами даних та клієнти зовнішніх API як інших сервісів, так і сторонніх інструментів. Даний шар абстрагує деталі технічної реалізації взаємодії з зовнішніми ресурсами, надаючи вищим шарам абстракцію для роботи з даними.

Центральне місце в архітектурі посідає шар «Domain», який інкапсулює всю бізнес-логіку системи та правила предметної галузі. Цей шар складається з сервісів, що реалізують ключові операції системи: управління користувачами, обробку інформації про майно, логіку дошок та проміжну валідацію даних. Важливою особливістю є те, що даний шар оперує абстракціями, не залежачи напряму від конкретних технологій збереження даних чи комунікації. Завдяки цьому бізнес-логіка залишається стійкою до змін у технологічному стеку та може бути легко перевірена за допомогою автоматизованих тестів.

Верхнім шаром архітектури є «API», який надає точки входу до системи через різні протоколи та формати обміну даними. Тут реалізовано REST та gRPC обробники для синхронної HTTP-взаємодії, а також споживачі повідомлень з черг для асинхронної обробки. Цей шар відповідає за валідацію вхідних даних, перевірку сесії та трансформацію внутрішніх моделей у формати, що є зручними для використання клієнтськими застосунками.

Принципово важливою характеристикою обраної архітектури є односпрямованість залежностей: шари використовують інтерфейси та не залежать один від одного напряму, в той час як принцип інверсії залежностей відбувається прямолінійно, щоб вищі шари використовували

нижчі, але не навпаки. Така організація забезпечує можливість заміни компонентів одного шару без впливу на інші, що суттєво підвищує гнучкість системи, спрощує її еволюційний розвиток, мінімізує зв'язаність та забезпечує можливість незалежного тестування кожного компонента.

3.3.3. Забезпечення безпеки, цілісності та стійкості застосунку

Для створення надійної та захищеної системи впроваджено комплексний підхід до забезпечення безпеки на всіх рівнях архітектури. Центральним елементом цієї стратегії є API Gateway, який виступає єдиною точкою входу для всіх зовнішніх запитів до системи. Цей компонент використовує авторизатор, який здійснює автентифікацію кожного запиту перед його подальшою маршрутизацією до відповідних внутрішніх служб. Додатково, для запобігання зловмисним атакам для відмови системи маршрутизатор реалізовує обмежувач частоти запитів, який контролює кількість звернень від окремих користувачів або IP-адрес за певний проміжок часу.

Для захисту внутрішнього мережевого трафіку між компонентами системи застосовано VPC (рис. 3.3), що створює ізольоване мережеве середовище. Таким чином, внутрішній обмін даними між службами відбувається в захищеному просторі, недоступному для зовнішніх спостерігачів. Це рішення суттєво зменшує вразливість системи до мережевих атак та несанкціонованого доступу до внутрішніх інтерфейсів сервісів.

Особлива увага приділяється забезпеченню стійкості системи до непередбачуваних ситуацій. Усі компоненти розроблено з дотриманням принципів Graceful Shutdown, що дозволяє службам завершувати поточні операції, зберігати стан та вивільняти ресурси під час планових оновлень, перезапусків або непередбачених вимкнень. Це гарантує, що жодна операція не буде перервана неналежним чином, а дані користувачів залишаться цілісними навіть за складних обставин.

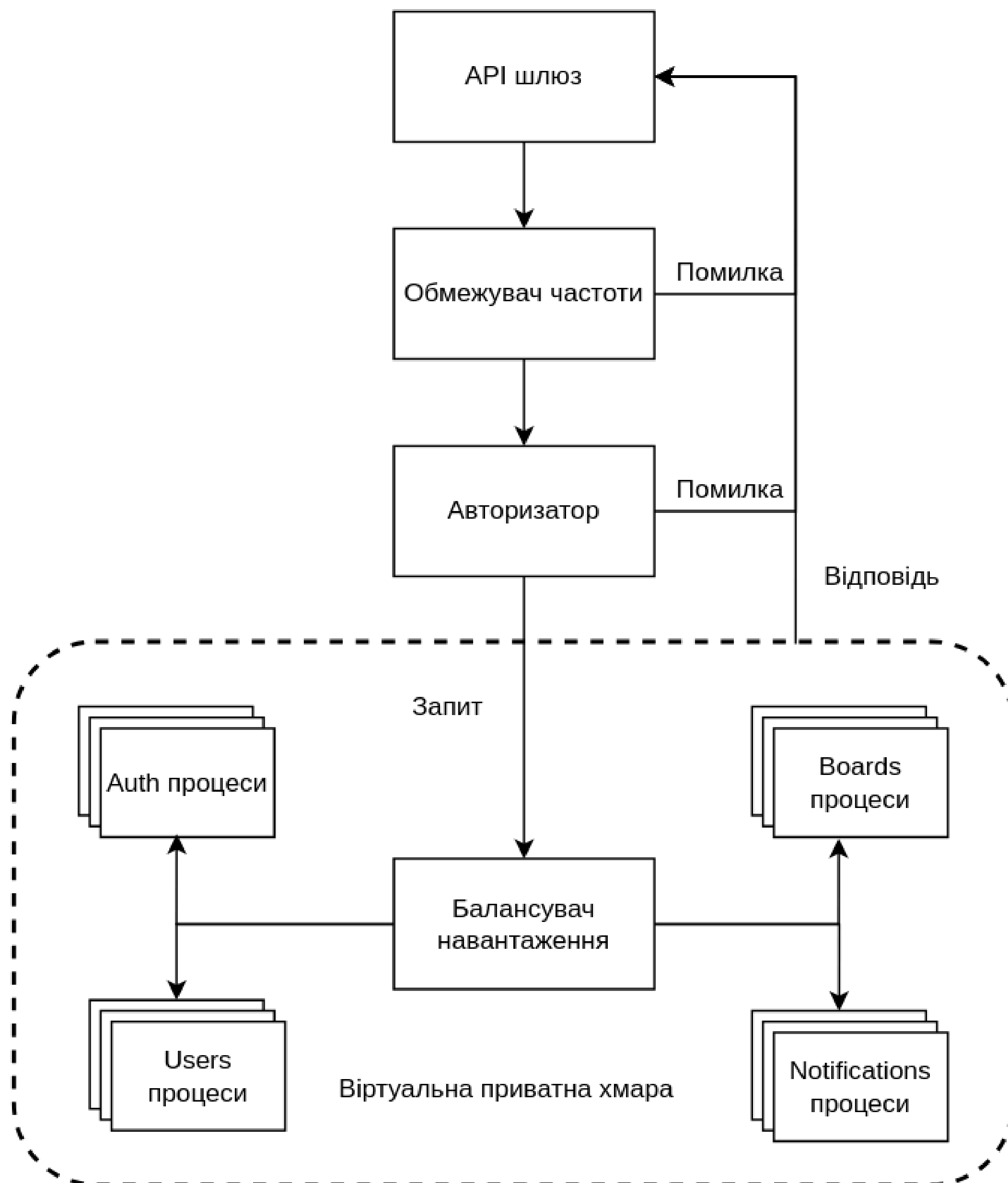


Рис. 3.3. Схема адресації зовнішніх запитів

Для забезпечення гнучкої продуктивності та оптимального використання обчислювальних ресурсів впроваджено автоматичне масштабування контейнерів у хмарній інфраструктурі за допомогою AWS Elastic Container Service Tasks та Auto Scaling Group [11]. Система постійно відстежує навантаження на кожну службу та автоматично збільшує або зменшує кількість запущених екземплярів відповідно до встановлених метрик. Це дозволяє ефективно обслуговувати піки навантаження без надмірного резервування ресурсів у періоди низької активності.

Невід'ємною частиною цієї архітектури є перевірка справності, які регулярно оцінюють стан кожного компонента системи. При виявленні несправності запускаються процедури самовідновлення, які можуть включати перезапуск проблемного екземпляру служби або перенаправлення трафіку на резервні копії.

Цілісність даних у розподіленому середовищі забезпечується за допомогою каскадних оновлень між мікросервісами. Для реалізації цього підходу використовується Kafka, а саме розподілені черги – Topics, що зберігають події з фактором реплікації [12] між декількома серверами для уникнення втрати даних. Кожна значуща зміна даних публікується як подія у відповідній тематичній черзі, а зацікавлені сервіси підписуються на ці канали для синхронізації своїх локальних сховищ даних. Такий асинхронний підхід до поширення змін забезпечує високу доступність системи навіть у випадку тимчасової недоступності окремих компонентів.

Для відстеження всіх дій у системі та швидкого виявлення нестандартної поведінки впроваджено багаторівневу систему логування. Усі значущі події, включаючи запити користувачів, зміни даних та системні операції, фіксуються в централізованому сховищі журналів. Це дозволяє проводити ретроспективний аналіз проблем, виявляти потенційні вразливості та оперативно реагувати на підозрілу активність.

Завдяки комплексному впровадженню цих заходів безпеки та стійкості, створена система не лише захищає дані користувачів від несанкціонованого доступу, але й забезпечує високу доступність сервісу навіть за складних умов експлуатації.

3.4. Структура бази даних

Основою для збереження та опрацювання інформації в системі є реляційна система керування базами даних PostgreSQL. Її вибір зумовлений потребою в надійному збереженні структурованих даних із забезпеченням цілісності зв'язків між різними категоріями інформації.

PostgreSQL забезпечує підтримку складних запитів, транзакційну обробку та контроль доступу до даних, що критично важливо для забезпечення надійної роботи з інформацією користувачів. Ця система дозволяє ефективно зберігати та обробляти великі обсяги структурованої інформації, підтримуючи при цьому сувору типізацію даних та забезпечуючи високу продуктивність складних запитів.

Для збереження та аналізу логів діяльності системи використовується спеціалізоване рішення Elasticsearch. Цей інструмент оптимізовано для швидкого пошуку та аналізу великих обсягів неструктурованих даних, що робить його ідеальним для роботи з журналами подій.

3.4.1. Опис таблиць та зв'язків між ними

Важливо зазначити, що оскільки система реалізовує мікросервісну архітектуру, кожен сервіс має власний екземпляр бази даних PostgreSQL. Такий підхід забезпечує ізоляцію даних та незалежність кожного сервісу, проте створює виклики щодо підтримки узгодженості даних між різними частинами системи. Для забезпечення цілісності інформації в розподіленій системі впроваджено механізми асинхронної комунікації через черги повідомлень, що дозволяє надійно синхронізувати дані між окремими екземплярами баз даних без інтеграції прямих залежностей між сервісами.

База даних сервісу аутентифікації (рис. 3.4) також реалізовує лише одну таблицю, адже наразі тільки була інтегрована система перевірки паролю.

crredentials	
PK	**user_id uuid NOT NULL**
	password_hash nvarchar(255) NOT NULL
	password_salt nvarchar(255) NOT NULL
	created_at timestamp NOT NULL
	updated_at timestamp NOT NULL

Рис. 3.4. Таблиця сервісу аутентифікації

Таблиця credentials містить дані для автентифікації користувачів. Опис полів таблиці:

- user_id – ідентифікатор користувача, зовнішній ключ до таблиці.
- password_hash – хеш паролю користувача.
- password_salt – сіль, що використовується для посилення безпеки хешування паролю.
- created_at – дата та час створення облікового запису.
- updated_at – дата та час останньої зміни облікових даних.

База даних сервісу користувачів (рис. 3.5) реалізовує лише одну таблицю, якої достатньо для збереження персональних даних.

users	
PK	**id uuid NOT NULL**
	email nvarchar(255) NOT NULL UNIQUE
	nickname nvarchar(255) NOT NULL UNIQUE
	first_name nvarchar(255) NOT NULL
	last_name nvarchar(255) NOT NULL
	avatar_url text NULL
	created_at timestamp NOT NULL
	updated_at timestamp NOT NULL

Рис. 3.5. База даних сервісу користувачів

Таблиця users зберігає основну інформацію про зареєстрованих користувачів системи. Опис полів таблиці:

- id – унікальний ідентифікатор користувача.
- email – електронна пошта користувача.
- nickname – унікальне ім’я користувача в системі.
- first_name – ім’я користувача.
- last_name – прізвище користувача.
- avatar_url – посилання на зображення профілю користувача.
- created_at – дата та час створення облікового запису.
- updated_at – дата та час останнього оновлення профілю.

База даних сервісу дошок (рис. 3.6) реалізовує п’ять таблиць для роботи з майном, його зображеннями, учасниками та історією.

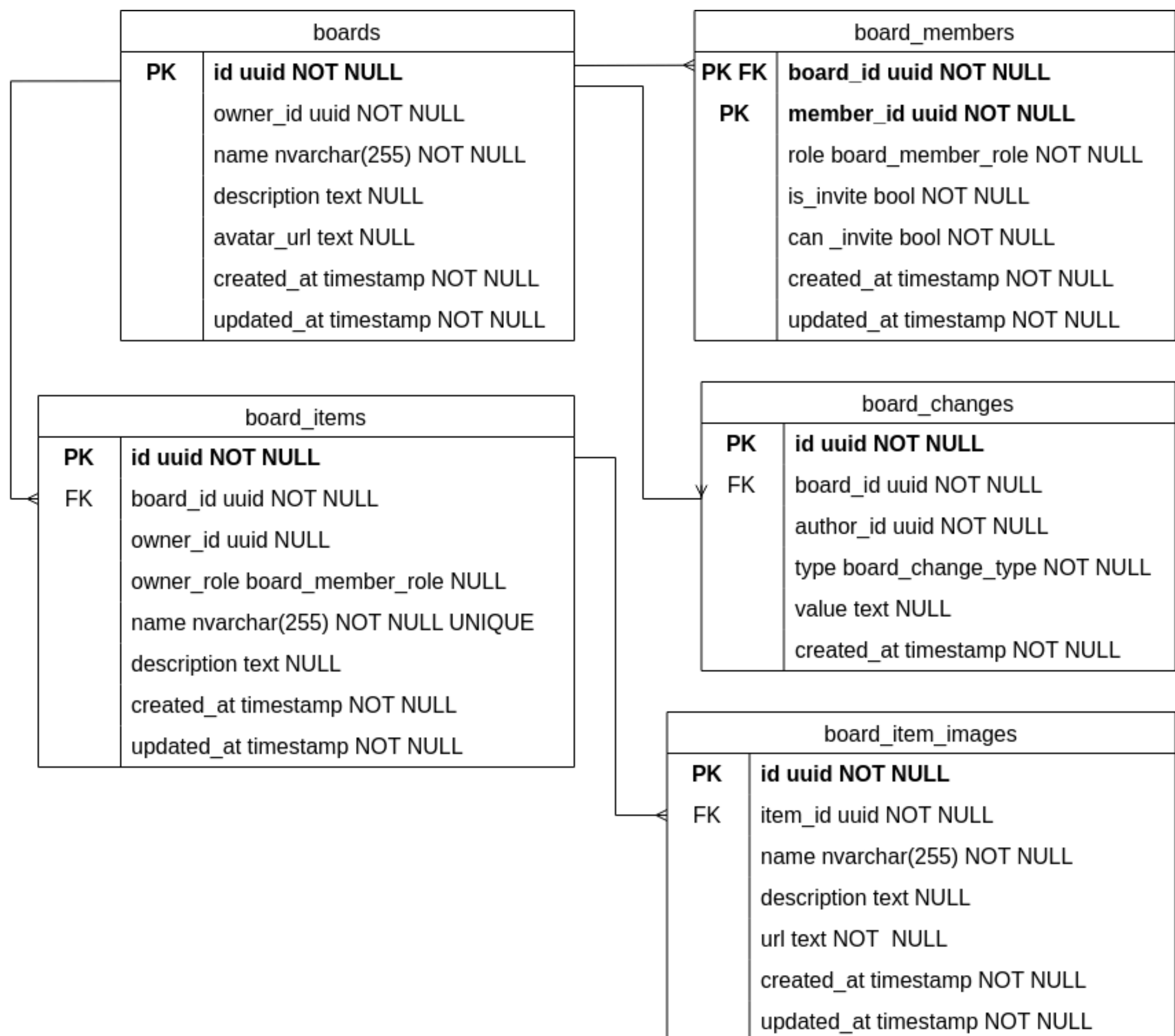


Рис. 3.6. Таблиці сервісу дошок

Таблиця boards зберігає загальну інформацію про дошки приміщень. Опис полів таблиці:

- id – ідентифікатор дошки.
- owner_id – ідентифікатор власника дошки.
- name – назва дошки приміщення.
- description – детальний опис дошки.
- avatar_url – посилання на загальне зображення дошки.
- created_at – дата та час створення дошки.
- updated_at – дата та час останнього оновлення інформації дошки.

Таблиця board_members визначає учасників каталогів та їхні права доступу. Опис полів таблиці:

- board_id – ідентифікатор дошки.
- member_id – ідентифікатор користувача, учасника дошки.
- role – роль користувача в дошці.
- is_invite – прапорець, що вказує чи учасник приймає запрошення.
- created_at – дата та час додавання користувача до дошки.
- updated_at – дата та час останньої зміни даних учасника.

Таблиця board_items містить інформацію про майно в дошках. Опис полів таблиці:

- id – унікальний ідентифікатор майна.
- board_id – ідентифікатор дошки, якому належить майно.
- owner_id – ідентифікатор власника майна.
- owner_role – роль власника майна.
- name – назва майна.
- description – опис майна.
- created_at – дата та час створення запису про майно.
- updated_at – дата та час останнього оновлення інформації про майно.

Таблиця board_changes містить історію змін дошки. Опис полів таблиці:

- id – унікальний ідентифікатор запису про зміну.
- board_id – ідентифікатор дошки, у якій відбулася зміна.
- author_id – ідентифікатор користувача, який здійснив зміну.
- type – тип зміни.
- value_from – попереднє значення зміненого поля.
- value_to – нове значення зміненого поля.
- created_at – дата та час внесення зміни.

Таблиця board_item_images зберігає зображення майна. Опис полів таблиці:

- id – унікальний ідентифікатор зображення.
- item_id – ідентифікатор майна, якому належить зображення.
- name – назва зображення.
- description – опис зображення.
- url – посилання на зображення.
- created_at – дата та час додавання зображення.
- updated_at – дата та час останнього оновлення інформації про зображення.

База даних сервісу сповіщень (рис. 3.5) реалізовує чотири таблиці для роботи з налаштуваннями, отримувачами та прив'язкою до дошок.

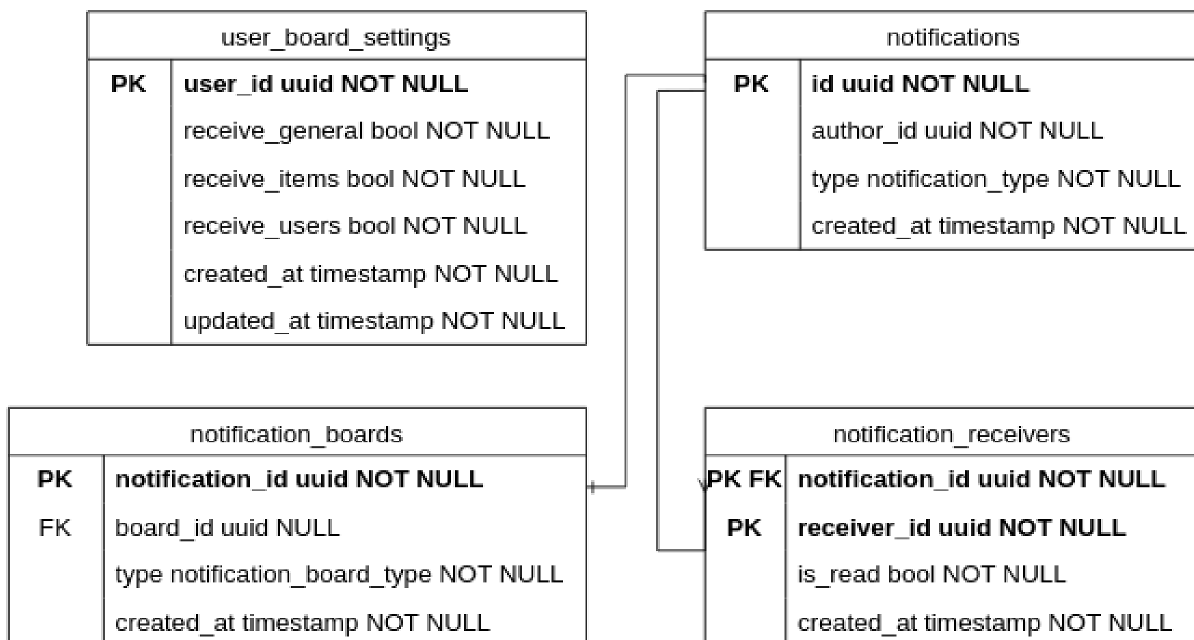


Рис. 3.7. Таблиці сервісу сповіщень

Таблиця user_board_settings зберігає налаштування сповіщень дошок користувача. Опис полів таблиці users:

- user_id – ідентифікатор користувача.
- receive_general – прапорець отримання загальних сповіщень дошки.
- receive_items – прапорець отримання сповіщень про майно дошки.

- receive_users – прапорець отримання сповіщень про учасників дошки.
- created_at – дата та час створення налаштувань.
- updated_at – дата та час останнього оновлення налаштувань.

Таблиця notifications містить інформацію про сповіщення. Опис полів таблиці users:

- id – унікальний ідентифікатор сповіщення.
- author_id – ідентифікатор користувача, який спричинив подію.
- type – тип сповіщення.
- created_at – дата та час створення сповіщення.

Таблиця notification_boards пов'язує сповіщення з дошками. Опис полів таблиці users:

- notification_id – ідентифікатор сповіщення.
- board_id – ідентифікатор дошки, якого стосується сповіщення.
- type – тип сповіщення дошки.
- created_at – дата та час створення зв'язку.

Таблиця notification_receivers визначає отримувачів сповіщень. Опис полів таблиці users:

- notification_id – ідентифікатор сповіщення.
- receiver_id – ідентифікатор користувача-отримувача.
- is_read – прапорець, що вказує чи було прочитано сповіщення.
- created_at – дата та час надсилання сповіщення.
- updated_at – дата та час останнього оновлення інформації сповіщення.

3.4.2. Особливості реалізації логування

Ефективне відстеження та аналіз подій в мікросервісній системі вимагає комплексного підходу до збору, обробки та візуалізації журнальних записів. Важливим методологічним рішенням стало впровадження логування за допомогою снепшотів – знімків стану. Цей

підхід передбачає запис не лише самої події, але й повного контексту її виникнення, включаючи стан сутності у результаті події. На відміну від класичного текстового журналювання, знімки забезпечують значно вищу інформативність записів, дозволяючи відтворити повну картину того, що відбувалося в системі в конкретний момент часу. Це критично важливо для розуміння причинно-наслідкових зв’язків між різними подіями, особливо коли йдеться про складні операції, що розподілені між декількома мікросервісами.

Для технічної реалізації системи логування впроваджено сучасний стек технологій ELK [13] – Elasticsearch, Logstash, Kibana (рис. 3.8), що доповнений брокером повідомлень Kafka для надійної передачі даних.

Процес збору та аналізу журнальних записів організовано таким чином: кожен мікросервіс при виникненні подій (запити, мутації або помилки) формує структуровані записи у форматі JSON та публікує їх у відповідному тематичному каналі Kafka. Використання Kafka як проміжної ланки забезпечує буферизацію даних, стійкість до тимчасових відмов компонентів системи та можливість горизонтального масштабування обробки журналів при зростанні навантаження.

З черги повідомлень Kafka записи надходять до набору спеціалізованих пайплайнів Logstash. Важливою особливістю нашої реалізації є розділення пайплайнів за типами даних, що дозволяє застосовувати специфічні правила обробки для різних сутностей мікросервісів. Такий підхід забезпечує оптимальну обробку різнотипної інформації та її підготовку для ефективного зберігання та аналізу.

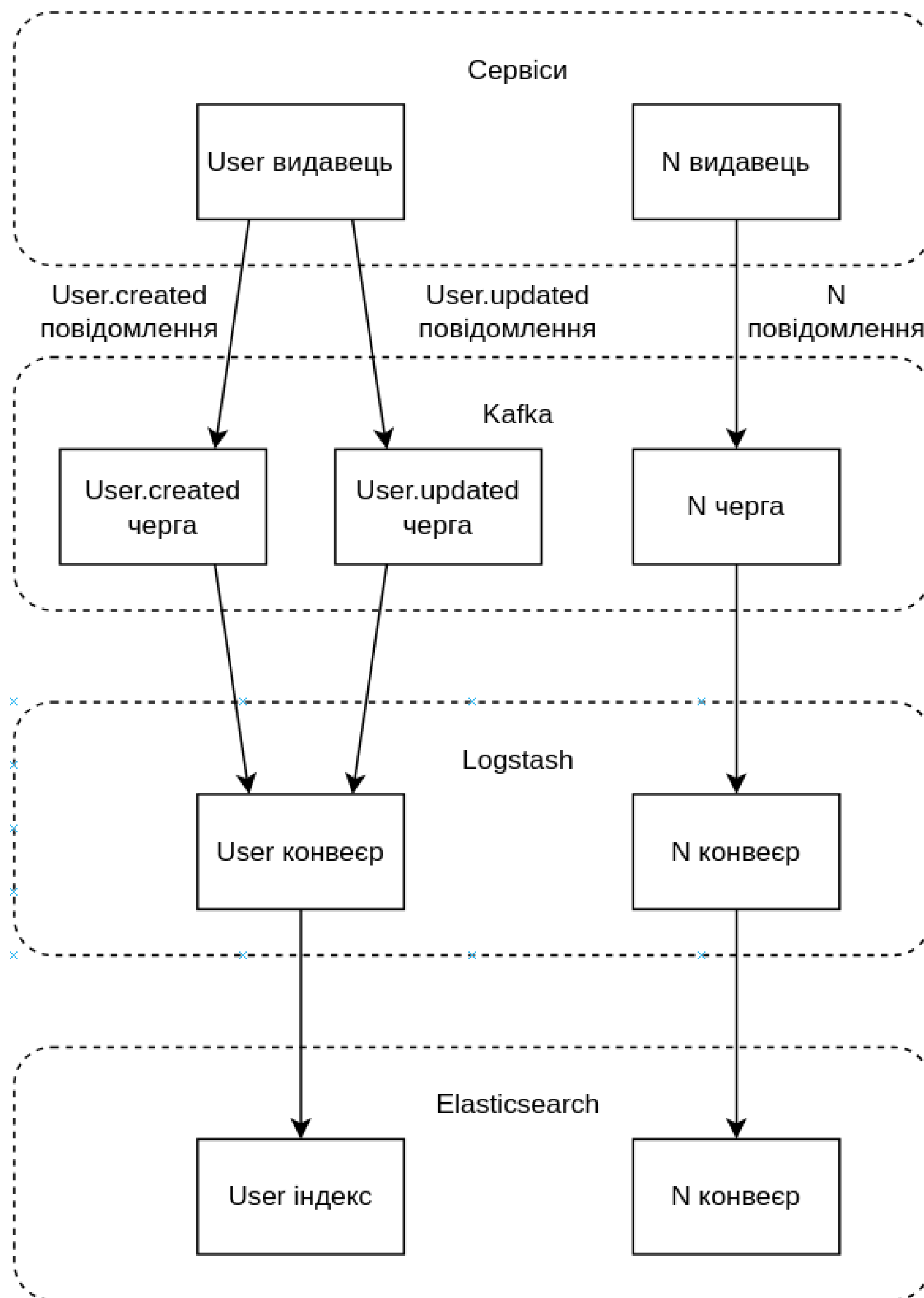


Рис. 3.8. Схема етапів логування

Оброблені та структуровані журнальні записи зберігаються в індексах Elasticsearch – розподіленій системі, оптимізованій для роботи з великими обсягами неструктурованих даних. Elasticsearch забезпечує вражаючу швидкість пошуку та аналізу, завдяки застосуванню інвертованих індексів та розподіленої архітектури. Система автоматично створює часові індекси для різних типів журналів, що дозволяє ефективно керувати життєвим циклом даних.

Завершальною ланкою в екосистемі логування є платформа візуалізації Kibana, яка надає інтуїтивно зрозумілий вебінтерфейс для взаємодії з даними, збереженими в Elasticsearch. Особлива цінність Kibana полягає в можливості створення різноманітних інформаційних панелей, адаптованих для різних сценаріїв використання: від моніторингу технічних показників до бізнес-аналітики. Інтерактивні діаграми, графіки та таблиці дозволяють швидко виявляти аномалії, відстежувати тренди та заглиблюватися в деталі конкретних подій. Особливо корисною є функціональність часових фільтрів, що дозволяє точно локалізувати проблемні ситуації та відстежувати ланцюжки пов'язаних подій через унікальні ідентифікатори.

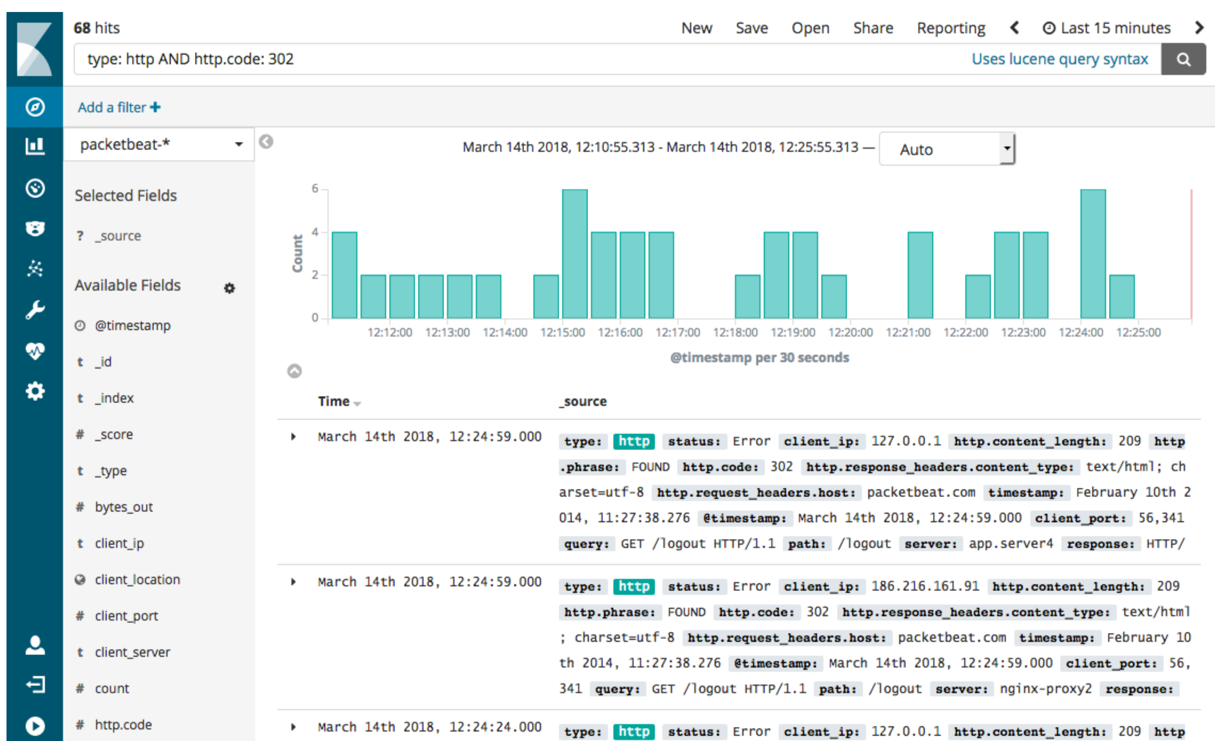


Рис. 3.9. Інтерфейс Kibana для перегляду індексів

Комплексне впровадження системи логування надає інструмент для оперативного виявлення та діагностики проблем, проактивного моніторингу стану системи та глибокого аналізу сценаріїв використання.

3.5. Особливості реалізації клієнтської частини застосунку

Клієнтська частина застосунку була реалізована з використанням фреймворку Next.js та реалізацією SSR. Цей вибір зумовлений низкою суттєвих переваг, які надає такий підхід порівняно з класичними SPA.

Серверне відображення забезпечує значно кращу початкову швидкість завантаження сторінки, оскільки користувач отримує вже готовий HTML-вміст замість порожньої розмітки, яка поступово заповнюється. Це суттєво покращує досвід користувача, особливо на мобільних пристроях або при повільному з’єднанні до інтернету. Також критичною перевагою є оптимізація для пошукових систем, адже пошукові системи отримують повноцінний HTML-контент під час індексації, що покращує видимість застосунку в результатах браузеру.

Важливим аспектом є також покращена безпека системи, оскільки частина логіки виконується на сервері, а не повністю відкрита у клієнтському коді. Фреймворк пропонує гібридну модель, яка дозволяє комбінувати переваги статичного генерування, серверного відображення та клієнтського відображення для різних частин застосунку залежно від їхніх потреб, що забезпечує оптимальний баланс між продуктивністю та інтерактивністю. Враховуючи потенційне використання застосунку, головним пріоритетом була розробка оптимального вигляду для мобільних пристроїв, проте сторінки будуть підлаштовуватися до будь-якого розміру екрану.

Відповідно до розробленої архітектури, застосунок структуровано за групами маршрутів відповідно до їхньої задачі.

Маршрути для роботи з обліковим записом:

- /login – сторінка аутентифікації користувача, що забезпечує безпечний вхід до системи та є публічною.
- /registration – сторінка для створення нового облікового запису в системі та також є публічною.

Маршрути для роботи з дошками:

- /board – основна сторінка з відображенням списку доступних користувачу дошок, доступна всім автентифікованим користувачам.
- /board/invitation – сторінка для перегляду та персональними управління запрошеннями до дошок від інших користувачів, також доступна всім автентифікованим користувачам.
- /board/[id] – основна сторінка дошки з відображенням списку майна та навігацією до інших сторінок, доступна учасникам дошки.
- /board/[id]/member – сторінка управління учасниками дошки та власною присутністю, також доступна учасникам дошки.
- /board/[id]/history – сторінка історії змін дошки та її майна, доступна учасникам дошки.

Маршрути для роботи з сповіщеннями:

- /notifications – сторінка для перегляду персональних сповіщень користувача, доступна всім автентифікованим користувачам.

Маршрути для роботи з налаштуваннями:

- /settings – сторінка для перегляду опцій наоаштувань, доступна всім автентифікованим користувачам.
- /settings/profile – сторінка для редагування особистих даних користувача та зміни паролю, доступна всім автентифікованим користувачам.
- /settings/notifications – сторінка для налаштування відображення персональних сповіщень, доступна всім автентифікованим користувачам.

Усі маршрути реалізовано з урахуванням принципів доступності інтерфейсу, забезпечуючи коректну роботу з клавіатурою, програмами зчитування з екрану та дотримання контрастності для людей з вадами зору. Завдяки можливостям фреймворку, система ефективно використовує кешування на рівні сторінок для підвищення швидкодії при повторних

відвідуваннях, а також впровадження реактивності для забезпечення якісного користувацького досвіду.

3.5. Висновки

Аналіз функціональних вимог дозволив визначити ключові можливості системи, серед яких управління дошками приміщень, облік майна, призначення опису, співпраця між учасниками та гнучка система сповіщень.

Розроблена серверна частина базується на сучасній мікросервісній архітектурі, що забезпечує оптимальну ізоляцію компонентів та можливість незалежного масштабування. Виділення сервісів Users Service, Auth Service, Boards Service, Notifications Service дозволило створити стійку до відмов систему з чітким розмежуванням відповідальності. Для забезпечення надійної комунікації між сервісами впроваджено гібридний підхід: синхронна взаємодія через HTTP для критичних операцій та асинхронна через брокери повідомлень для обміну подіями.

Особливу увагу приділено безпеці системи та захисту даних. Впроваджено комплексну систему аутентифікації з використанням централізованих посередників та переадресації, шифрування критичних даних та ізоляцію внутрішнього трафіку в рамках віртуальної приватної хмари. Контроль доступу реалізовано на всіх рівнях системи, забезпечуючи принцип найменших привілеїв для користувачів та сервісів. Проєктування з урахуванням потенційних відмов компонентів забезпечує стійкість системи до збоїв та мінімізацію простоїв.

Структура бази даних розроблена з урахуванням специфіки мікросервісної архітектури. Кожен сервіс використовує окремий екземпляр з розробленою для конкретних завдань схемою, що забезпечує високу продуктивність та масштабованість. Впроваджено механізми підтримки узгодженості даних між різними сервісами з використанням потоків подій. Для ефективного моніторингу та діагностики реалізовано комплексну

систему логування на базі стеку ELK з використанням снепшотів для журналювання повного контексту подій.

Клієнтська частина реалізована з використанням Next.js, що забезпечує оптимальний баланс між продуктивністю, інтерактивністю та пошуковою оптимізацією завдяки технології серверного відображення. Інтуїтивний інтерфейс користувача, розроблений з урахуванням сучасних принципів дизайну, забезпечує зручну взаємодію з системою на різних пристроях.

Важливим аспектом реалізації стало впровадження механізмів автоматичного масштабування та моніторингу працездатності системи. Використання контейнеризації дозволило ефективно управляти ресурсами та забезпечити стабільну роботу при змінному навантаженні, а система перевірки стану компонентів забезпечує швидке виявлення та усунення проблем. Додатково реалізовано принципи коректного завершення роботи, що гарантує збереження цілісності даних навіть при непередбачуваних ситуаціях або планових оновленнях системи.

4. АНАЛІЗ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Аналіз реалізованого застосунку

В результаті виконаної розробки програмного забезпечення було створено зручний вебзастосунок для фіксації орендованого майна, що дозволяє користувачам легко вести облік майна у приміщеннях та спільно керувати їх використанням.

Головною особливістю системи є дошки – інтерактивні сторінки приміщень, які дають можливість структуровано зберігати інформацію про майно. Для початку роботи з системою потрібно пройти просту процедуру реєстрації або входу, після чого користувач отримує доступ до повного функціональності застосунку.

На головній сторінці користувач бачить картки з доступними дошками приміщень. Перехід до конкретної дошки відкриває детальний перегляд її вмісту – тут можна побачити майно, додати нове, змінити опис існуючих або завантажити супутні фотографії.

Особливу цінність застосунку надає можливість командної роботи. Власник дошки може запросити колег чи партнерів до спільного використання, що особливо корисно для орендодавців та орендарів, які спільно ведуть облік майна. Система вчасно інформує всіх учасників про зміни через сповіщення – додавання нового майна, зміну їхнього опису або приєднання нових учасників.

На сторінці налаштувань кожен користувач може персоналізувати свій профіль та відфільтрувати отримання сповіщень. Це дозволяє налаштувати систему під індивідуальні потреби та робочі процеси.

Основними перевагами розробленого рішення є його інтуїтивність, швидкість, чіткий візуальний стиль та простота використання. На відміну від складних систем управління майном, цей застосунок не вимагає спеціальних навичок або тривалого навчання. Водночас він надає необхідні інструменти для ефективного обліку та управління орендованим майном.

Важливою особливістю є адаптивний дизайн, який забезпечує зручність роботи як на комп'ютері, так і на мобільних пристроях. Це дозволяє користувачам вести облік майна безпосередньо під час огляду приміщень, доповнюючи записи фотографіями в режимі реального часу.

Завдяки продуманій організації інформації та зручним інструментам пошуку й сортування, користувачі можуть швидко знаходити потрібне майно навіть у великих дошках. Система сповіщень дозволяє оперативно реагувати на зміни та підтримувати актуальність даних.

4.2. Тестування реалізованої серверної частини застосунку

У процесі розробки застосунку було впроваджено багаторівневу стратегію перевірки працездатності, що охоплювала як серверні компоненти, так і взаємодію між ними. Для проведення випробувань серверних компонентів було використано двоетапний підхід, що поєднував автоматизовані перевірки бізнес-логіки та ручне тестування маршрутів через спеціалізовані інструменти.

Було розроблено набір автоматизованих тестів для перевірки доменного шару бізнес-логіки. Ці тести дозволили ізольовано запевнитися у правильності роботи основних алгоритмів та бізнес-правил системи без залежності від інфраструктурних компонентів, таких як бази даних або зовнішні сервіси.

Автоматизоване тестування доменного шару було реалізовано за наступними принципами:

- модульний Black Box підхід, при якому кожен сервіс тестувався окремо з використанням заглушок для зовнішніх залежностей;
- охоплення різноманітних сценаріїв використання, включаючи як успішні операції, так і обробку помилкових ситуацій;
- перевірка дотримання бізнес-правил та обмежень, визначених у вимогах до системи;

- тестування граничних випадків для виявлення потенційних проблем з обробкою нестандартних даних.

Критично важливим елементом процесу розробки стала інтеграція автоматизованих тестів у пайплайни безперервної інтеграції та доставки. При кожному внесенні змін до репозиторію автоматично запускався набір тестів, що перевіряв не лише нову функціональність, але й відсутність регресій у вже готових частинах сервісу. Система автоматично блокувала інтеграцію коду, що не проходив тести, запобігаючи потраплянню дефектів до основної гілки розробки та, як наслідок, до середовища експлуатації.

Для зручного ручного тестування API-інтерфейсів було створено колекцію в інструменті Postman. Ця колекція структурована за функціональними модулями системи та містить готові шаблони запитів для всіх доступних операцій. Кожен запит супроводжувався детальним описом його призначення, очікуваних вхідних параметрів та результатів виконання. Використання цього середовища дозволило не лише швидко перевіряти окремі точки входу API, але й створювати складні сценарії тестування з послідовним виконанням взаємопов’язаних операцій.

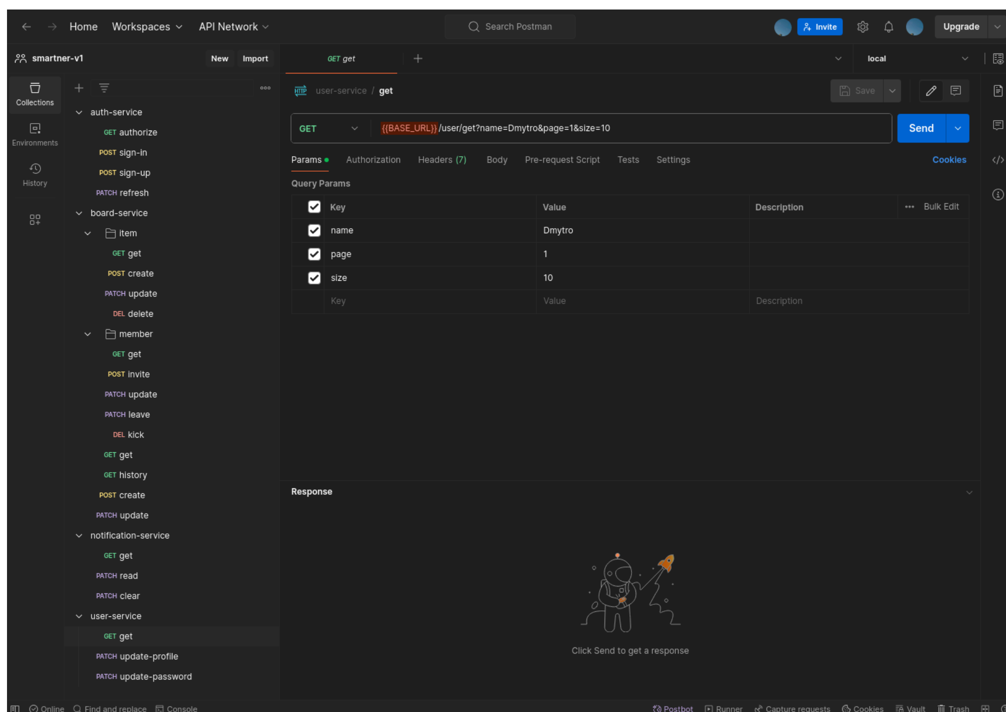


Рис. 4.1. Колекція Postman сервісу користувачів

Процес розробки колекції Postman включав наступні етапи:

1. Визначення набору ключових сценаріїв використання для кожного мікросервісу.
2. Додавання динамічних змінних середовища, наприклад секретні токени та ключі.
3. Підготовка тестових даних, що охоплюють як стандартні, так і граничні випадки.
4. Створення API маршрутів колекцій та документування даних.

Після завершення розробки було проведено фінальне комплексне тестування, яке підтвердило готовність системи до використання. Результати тестування показали, що виявлені раніше дефекти були усунені, а функціональність системи повністю відповідає поставленим вимогам.

4.3. Тестування реалізованої клієнтської частини застосунку

При тестуванні користувацького інтерфейсу застосунку основним підходом стала перевірка повних сценаріїв використання, що відображають реальну взаємодію користувача з продуктом.

Тестування відбувалося від імені різних типів користувачів – власників приміщень, орендарів, орендодавців – щоб переконатися, що інтерфейс однаково добре обслуговує потреби всіх цільових груп. При тестуванні особлива увага приділялася інтуїтивності інтерфейсу, а саме наскільки легко новий користувач може виконати типові завдання без додаткових інструкцій.

Для забезпечення якості на різних пристроях, кожен сценарій перевірявся як на настільних комп’ютерах, так і на мобільних пристроях різних розмірів. Це дозволило виявити проблеми з адаптивністю та зручністю використання на екранах різних форматів.

У процесі тестування документувалися не лише технічні дефекти, але й моменти, що викликали труднощі або незручності при використанні інтерфейсу. Такий підхід дозволив виявити неочевидні проблеми з

користувацьким досвідом та оптимізувати інтерфейс для більш інтуїтивного використання.

Процес ручного тестування користувацьких інтерфейсів включав наступні етапи:

1. Планування – визначалась функціональна дія та її межі, що будуть тестуватися.
2. Створення тестових сценаріїв – один з прикладів, що був розроблений для перевірки процесу авторизації користувача, наведено в табл. 4.1.
3. Виконання тестових випадків – відтворення сценаріїв на розробленому програмному забезпеченні.
4. Аналіз помилок – аналіз результату при виконанні тестових випадків та подальше виправлення помилкової поведінки застосунку.
5. Повторне тестування – тестові випадки, які попередньо викликали помилки, виконувались повторно. Також враховувалися успішно виконані тестові випадки, на які опосередковано вплинути здійснені зміни.

Таблиця 4.1

Тестові випадки процесу авторизації користувача

Номер тестового випадку	Умови тестового випадку	Очікуваний результат
Т-1	Перебуваючи на сторінці авторизації, користувач вводить пошту та пароль існуючого облікового запису й натискає на кнопку підтвердження.	Застосунок переадресовує користувача на головну сторінка застосунку.

Продовження табл. 4.1

Т-2	Перебуваючи на сторінці авторизації, користувач вводить пошту та пароль, що не збігаються з жодним обліковим записом, і натискає на кнопку підтвердження.	Якщо обліковий запис з введеною поштою не існує – виводиться повідомлення про відсутність знайденого користувача. Якщо обліковий запис з введеною поштою існує, проте не збігається пароль – виводиться повідомлення про неправильний пароль.
Т-3	Перебуваючи на сторінці авторизації, користувач залишає поля пошти та паролю порожніми й натискає на кнопку підтвердження.	Виводиться повідомлення про необхідність заповнення полів.

Завдяки сценарному підходу до тестування вдалося створити інтерфейс, який ефективно підтримує основні робочі процеси користувачів та забезпечує приємний досвід взаємодії з системою інвентаризації орендованого майна.

4.4. Рекомендації щодо подальшого вдосконалення

Розроблений вебзастосунок демонструє значний потенціал для подальшого вдосконалення та розширення функціональності. Можна виділити ряд перспективних напрямків розвитку продукту.

Першочерговим вдосконаленням може стати впровадження SSE для забезпечення оновлення даних у режимі реального часу. Це дозволить користувачам миттєво бачити зміни, внесені іншими учасниками дошки – додавання нових предметів, зміни опису або коментарі – без необхідності оновлювати сторінку. Така функціональність суттєво покращить досвід співпраці та підвищить ефективність використання системи в команді.

Важливим кроком у розвитку проєкту може стати впровадження монетизації через систему тарифних планів. Базовий план міг би

залишатися безкоштовним із певними обмеженнями, тоді як підписка відкривала б доступ до розширених функцій. Для корпоративних клієнтів варто передбачити окремий тарифний план з можливістю інтеграції з внутрішніми системами та розширеними налаштуваннями безпеки.

Важливим розширенням функціональності може стати інтеграція комунікаційних можливостей безпосередньо в дошки приміщень. Реалізація вбудованої системи чатів для кожної дошки дозволить учасникам обговорювати питання, пов’язані з конкретними предметами майна, не покидаючи контексту роботи. Це створить зручний спосіб звернути увагу колег на конкретне майно.

Перспективним напрямком є також розробка інструментів аналітики та звітності. Додавання можливості генерувати детальні звіти про стан майна, історію змін та прогнози щодо необхідності заміни майна надасть користувачам цінну інформацію для прийняття управлінських рішень. Візуалізація даних через графіки та діаграми підвищить наочність представлення інформації.

Для задоволення потреб міжнародних користувачів доцільно впровадити багатомовну підтримку інтерфейсу та можливість працювати з різними валютами при додаванні функціональності оцінки вартості майна. Це значно розширить потенційну аудиторію продукту та збільшить його глобальну конкурентоспроможність.

З точки зору технічного вдосконалення, інтеграція функцій офлайн режиму дозволить користувачам продовжувати роботу навіть за відсутності стабільного з’єднання до інтернету, з подальшою синхронізацією змін при відновленні доступу до мережі.

4.5. Висновки

Було детально представлено розроблений застосунок з акцентом на його функціональні можливості та архітектурні рішення. Описано основні компоненти програмного рішення, взаємозв’язки між ними та принципи

їхньої роботи. Система успішно реалізує весь необхідний функціональність для документування майна орендованих приміщень та забезпечує зручну співпрацю між користувачами.

Важливим етапом розробки стало комплексне тестування всіх компонентів системи. Завдяки впровадженню практик ручного тестування інтерфейсу на основі користувацьких сценаріїв, а також автоматизованих тестів для перевірки серверної логіки, вдалося досягти високого рівня якості та надійності кінцевого продукту. Особливу цінність має інтеграція тестування в процес безперервної розробки, що дозволило ефективно виявляти та усувати проблеми на ранніх етапах.

Аналіз завершеного рішення дозволив визначити перспективні напрямки подальшого вдосконалення. Серед ключових можливостей для розвитку системи варто відзначити впровадження оновлень у реальному часі, розробку системи монетизації, інтеграцію вбудованих чатів для комунікації та розширення аналітичних можливостей. При подальшому розвитку згідно з визначеними рекомендаціями система має потенціал трансформуватися з базового інструменту в повноцінну платформу.

ВИСНОВКИ

Метою даного дипломного проєкту є покращення якості документування стану майна під час процесу орендування. Система надає комплексне рішення для опису майна приміщення, відстеження його стану та забезпечення ефективної співпраці між зацікавленими сторонами.

Початковим етапом розробки став аналіз існуючих програмних продуктів у сфері управління майном та документообігу, що дозволило виявити їхні обмеження та сформувати бачення конкурентоспроможного рішення. На основі цього аналізу було визначено ключові функціональні вимоги та спроєктовано архітектуру, яка забезпечує гнучкість, масштабованість та надійність системи.

Для реалізації проєкту було обрано сучасний технологічний стек, що оптимально відповідає поставленим завданням. Серверна частина розроблена з використанням мови програмування Go, яка забезпечує високу продуктивність обробки запитів та ефективне використання обчислювальних ресурсів. Клієнтська частина вебзастосунку реалізована за допомогою фреймворку Next.js з серверним відображенням, що дозволило досягти оптимального балансу між швидкістю завантаження сторінок, інтерактивністю та пошуковою оптимізацією. Для збереження даних застосовано PostgreSQL, яка забезпечує надійність та гнучкість при роботі зі структурованою інформацією.

Важливим компонентом розробки стала мікросервісна архітектура з чітким розподілом відповідальності між сервісами, що забезпечує незалежне масштабування та стійкість до відмов. Впроваджено комплексну систему безпеки з багаторівневою аутентифікацією та контролем доступу, а також механізми моніторингу та логування для оперативного виявлення та вирішення проблем.

Всебічне тестування розробленого застосунку включало як ручну перевірку користувацького інтерфейсу на основі типових сценаріїв

використання, так і автоматизоване тестування серверної логіки з інтеграцією в процес безперервної розробки. Це дозволило створити стабільний та надійний продукт з високою якістю користувацького досвіду.

Розроблене програмне забезпечення «Smartner» надає користувачам зручну функціональність для систематизації інформації про орендоване майно, контролю його стану та спільного доступу до даних. Інтуїтивно зрозумілий інтерфейс, гнучкі можливості налаштування та ефективна система сповіщень створюють комфортне середовище для щоденної роботи з майном.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

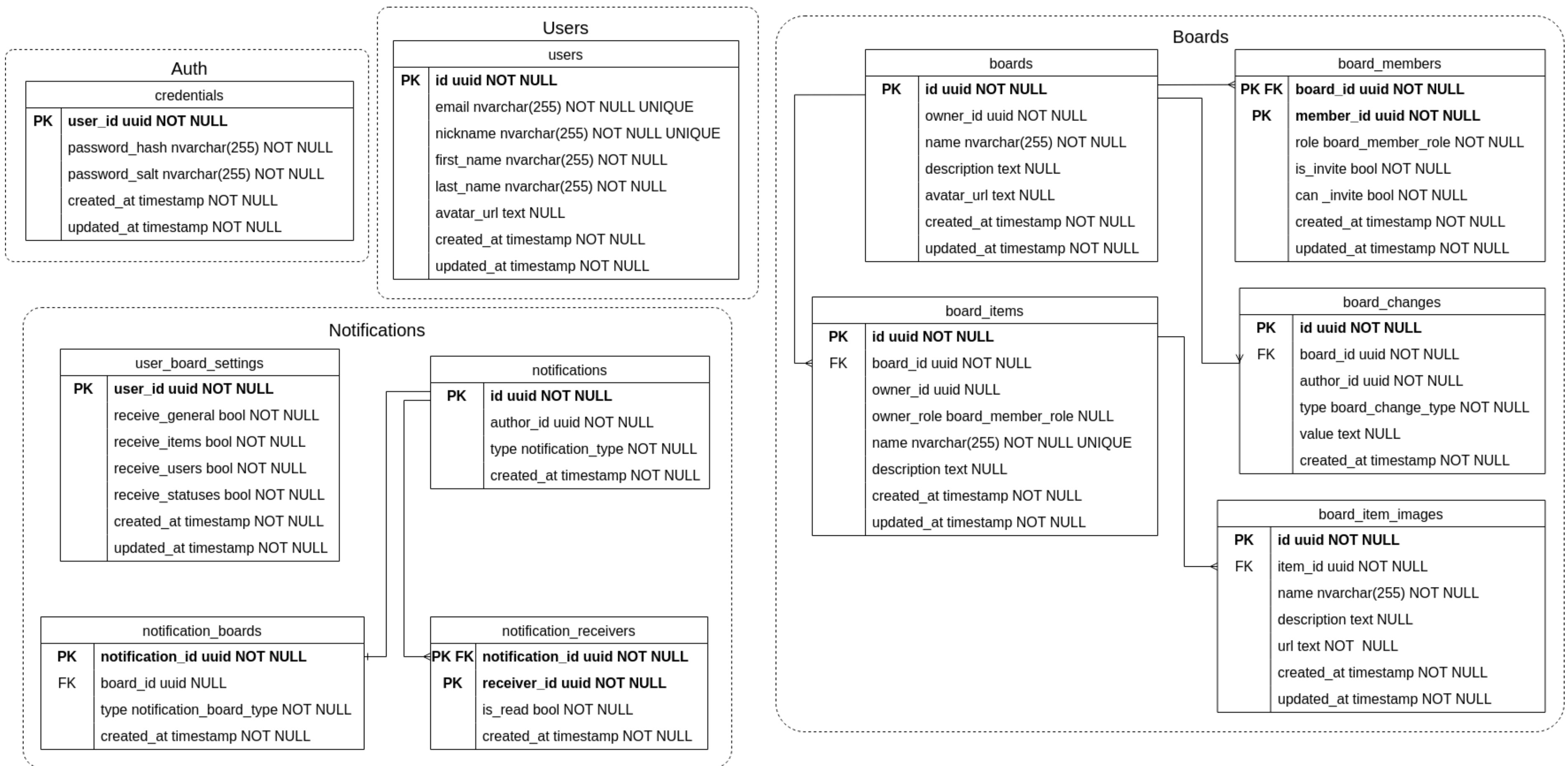
1. Digital Transformation in Property Documentation. [Електронний ресурс]. – Режим доступу: https://whatfix.com/blog/property-management-digital-transformation/ – Дата доступу: (16.03.2025).
2. Evernote vs. OneNote: Which note-taking app is best? [Електронний ресурс]. – Режим доступу: https://zapier.com/blog/evernote-vs-onenote/ – Дата доступу: (16.03.2025).
3. MRI Software, Property Tree Connect. [Електронний ресурс]. – Режим доступу: https://www.mrisoftware.com/uk/products/property-tree/ – Дата доступу: (15.03.2025).
4. Google Play, Rental Property Management Software. [Електронний ресурс]. – Режим доступу: https://play.google.com/store/apps/details?id=com.apps.PropertyManagerRentTracker – Дата доступу: (16.03.2025).
5. Why Go is Popular Right Now and Why I Started Learning Go as a Node.js Developer. [Електронний ресурс]. – Режим доступу: https://dev.to/mihailtd/why-go-is-popular-right-now-and-how-and-why-i-started-learning-go-as-a-nodejs-developer-2jcg – Дата доступу: (22.10.2024).
6. Microservices vs Monolith | Go & GoFr. [Електронний ресурс]. – Режим доступу: https://medium.com/@vipulrawat008/microservices-vs-monolith-go-gofr-d20c3b5f358a – Дата доступу: (10.06.2024).
7. A Deep Dive into Communication Styles for Microservices: REST vs. gRPC vs. Message Queues. [Електронний ресурс]. – Режим доступу: https://medium.com/@platform.engineers/a-deep-dive-into-communication-styles-for-microservices-rest-vs-grpc-vs-message-queues-ea72011173b3 – Дата доступу: (16.04.2025).

8. PostgreSQL чи MySQL. Як обрати оптимальну базу даних для вашого проєкту. [Електронний ресурс]. – https://dou.ua/forums/topic/51621 – Дата доступу: (17.12.2024).
9. Discovering Next.js: Why It’s My Go-To Framework for React Applications. [Електронний ресурс]. – https://adhithiravi.medium.com/discovering-next-js-why-its-my-go-to-framework-for-react-applications-dcb97fd81382 – (21.07.2023).
10. AWS Whitepaper. Microservices. [Електронний ресурс]. – https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/microservices.html – Дата доступу: (09.02.2025).
11. Automatically scale your Amazon ECS service. [Електронний ресурс]. – https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-auto-scaling.html – Дата доступу: (25.04.2025).
12. Kafka Replication and Committed Messages. [Електронний ресурс]. – https://docs.confluent.io/kafka/design/replication.html – Дата доступу: (07.04.2025).
13. What is Elastic Stack?. [Електронний ресурс]. – https://www.geeksforgeeks.org/what-is-elastic-stack-and-elasticsearch/ – Дата доступу: (07.11.2023).

ДОДАТКИ

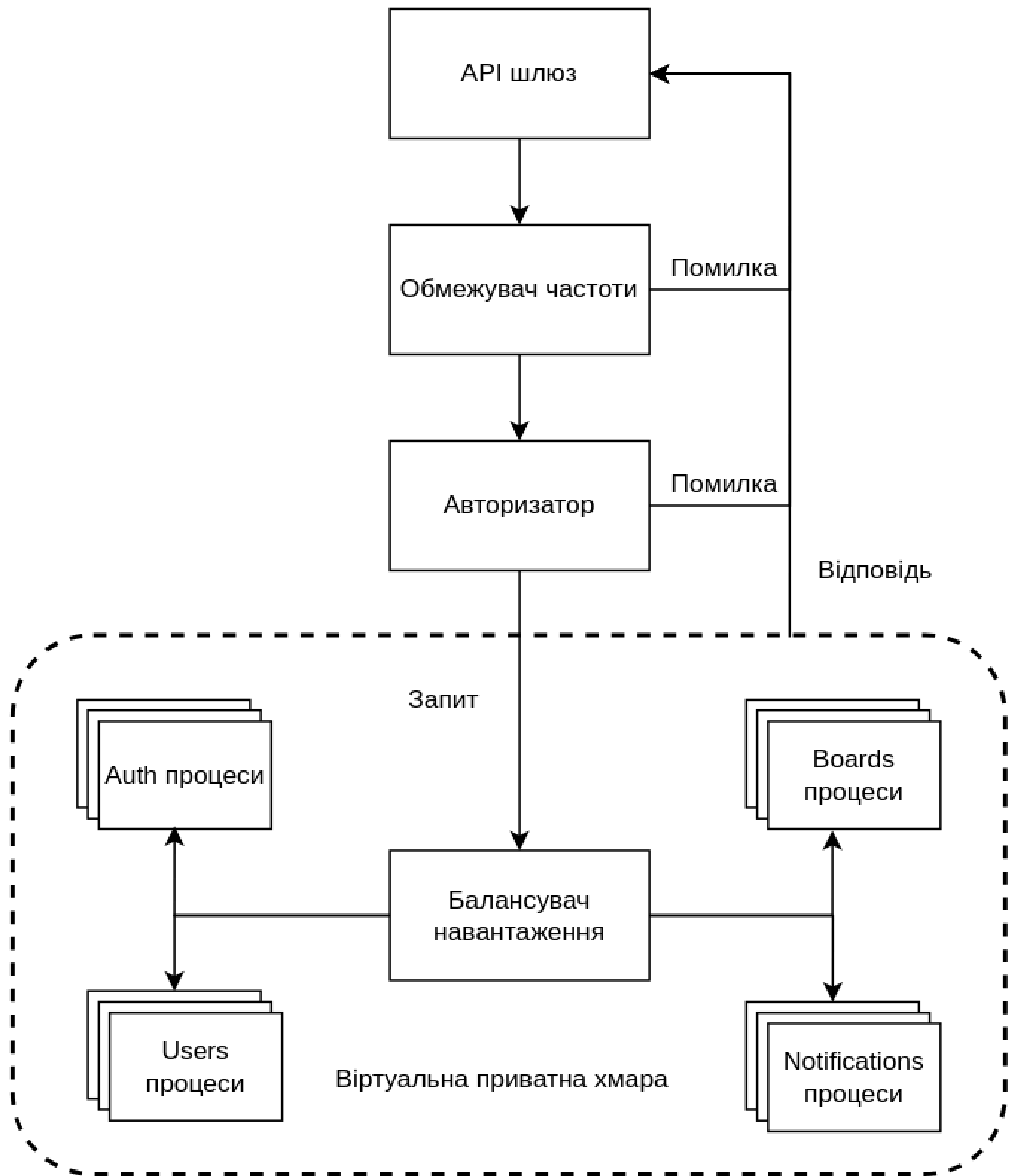
Додаток 1

Копії графічних елементів



ДП.045440-06-99

Програмне забезпечення для фіксування майна орендаря та орендодавця під час надання послуг з оренди. Структура бази даних. ER-діаграма



ДП.045440-07-99

Програмне забезпечення для фіксування майна орендаря та орендодавця під час надання послуг з оренди. Алгоритм адресації зовнішніх запитів. Блок-схема

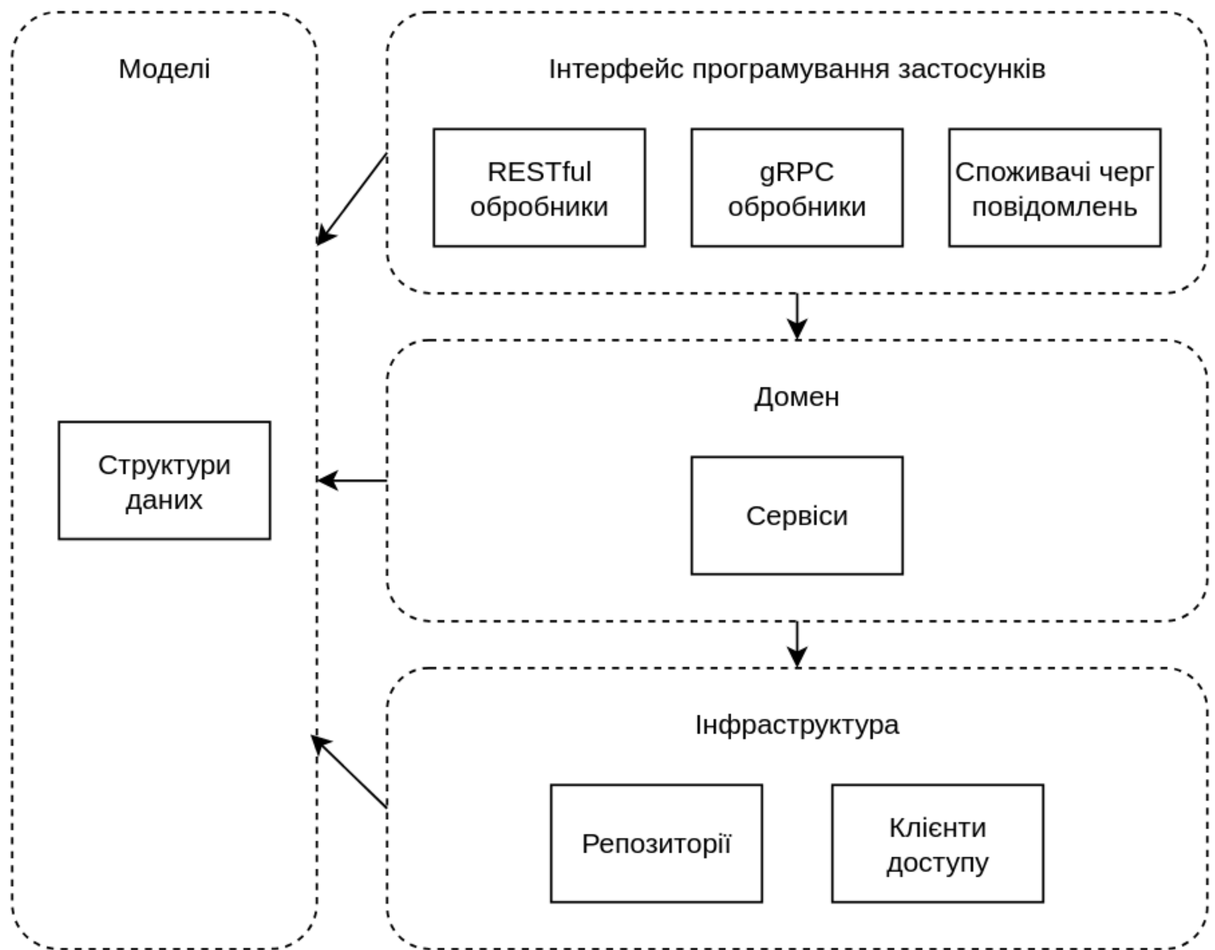


Схема архітектури кожного мікросервісу

Кучеренко Д.О., група КП-11

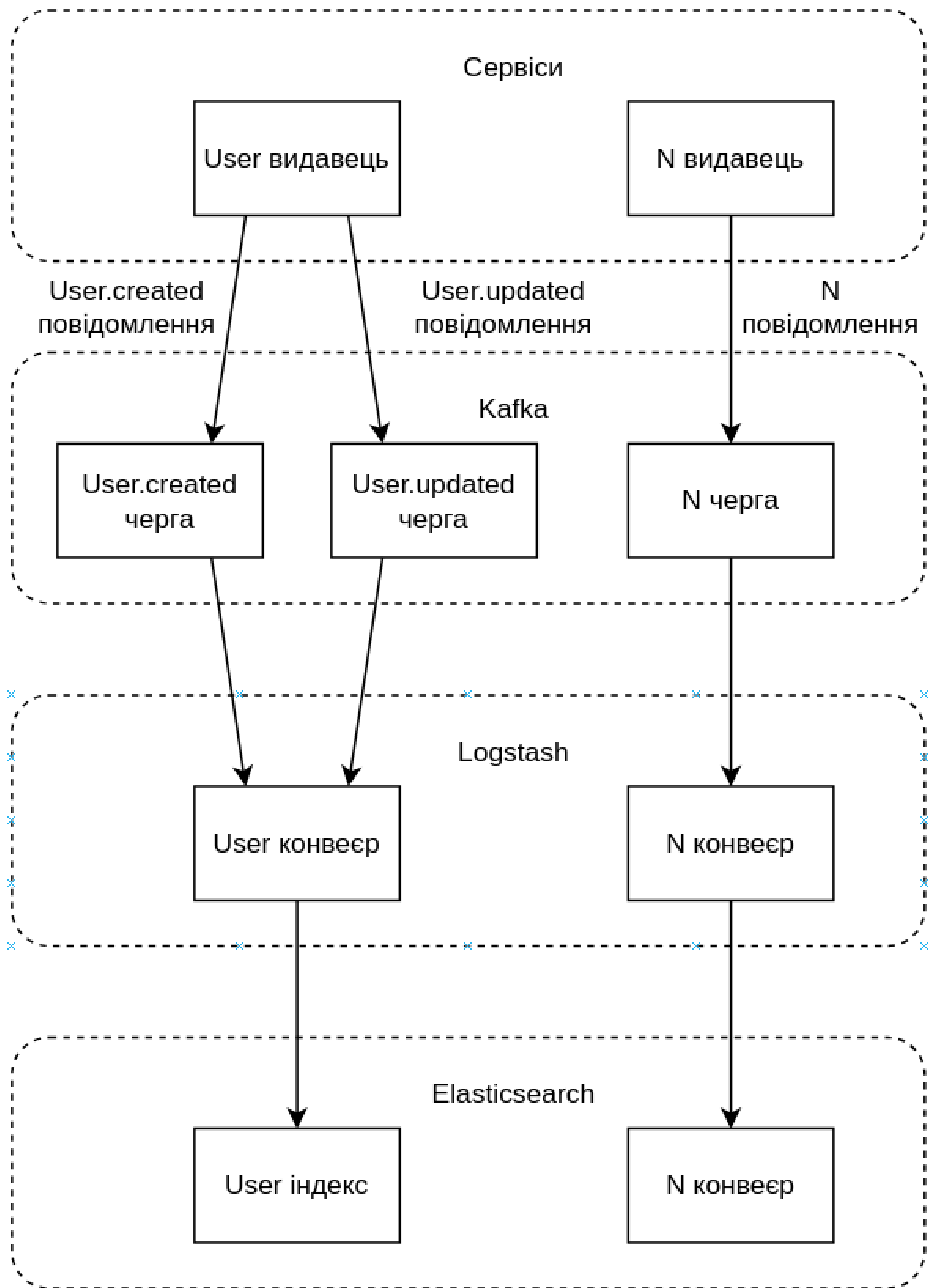


Схема етапів логування

Кучеренко Д.О., група КП-11

Додаток 2

Лістинг програми

Фрагменти коду інтерактивних елементів

```
import clsx from 'clsx';
import { Control, FieldValues, useController } from 'react-hook-form';

import { FieldFilteredPath } from '@/libs/types';

export type InputTextProps<T extends FieldValues> = {
  control: Control<T>;
  name: FieldFilteredPath<T, string | undefined>;
  type?: 'text' | 'email' | 'password';
  label?: string;
  placeholder?: string;
  className?: string;
  hideDetails?: boolean;
};

export function InputText<T extends FieldValues>({
  control,
  name,
  type = 'text',
  label,
  placeholder,
  className,
  hideDetails = false,
}: InputTextProps<T>): JSX.Element {
  const {
    field: { value, ...field },
    fieldState: { error },
  } = useController({ control, name });

  return (
    <label className={clsx('flex w-full flex-col gap-2', className)}>
      {label && <span className="pl-3 text-sm">{label}</span>}
      <input
        {...field}
        value={value ?? ''}
        type={type}
        name={label}
        placeholder={placeholder}
        className={clsx(
          error ? 'border-error' : 'border-common-border',
          'rounded-2xl border border-common-border p-3',
          'text-common bg-common-bg',
        )}
      />
      {!hideDetails && (
        <span
          className={clsx(
            !error && 'hidden',
            'ml-2 mt-0.5 text-xs font-thin text-error',
          )}
        >
          {error?.message ?? 'Placeholder'}
        </span>
      )}
    </label>
  );
}
```

```
import clsx from 'clsx';
import Link from 'next/link';
import { Route } from 'nextjs-routes';

import { Loader } from '@/components/atoms';

export type ButtonProps = {
  children: React.ReactNode;
  type?: 'submit' | 'button';
  to?: Route;
  name?: string;
  variant?: 'action' | 'text' | 'contained' | 'circle' | 'outlined';
  loading?: boolean;
  disabled?: boolean;
  fit?: boolean;
  className?: string;
  onClick?: () => void;
};

export function Button({
  children,
  type = 'button',
  variant = 'contained',
  to,
  name,
  loading = false,
  disabled = false,
  fit = false,
  className,
  onClick,
}: ButtonProps): JSX.Element {
  const getLinkWrapper = (children: React.ReactNode): JSX.Element => {
    if (to) {
      return (
        <Link
          className="flex h-full w-full items-center justify-center p-0"
          href={to}
          prefetch
        >
          {children}
        </Link>
      );
    }

    return <>{children}</>;
  };

  return (
    <button
      name={name}
      type={type}
      disabled={disabled || loading}
      onClick={onClick}
      className={clsx(
        className,
        'text-common flex items-center justify-center gap-1',
        {
          'w-full': !fit,
          'w-fit': fit,
          'rounded-3xl bg-primary px-6 py-2.5 text-lg
enabled:hover:brightness-hover disabled:bg-opacity-80':
            variant === 'contained',
          'rounded-3xl border border-primary bg-common-bg px-6 py-2.5 text-lg
text-primary enabled:hover:brightness-hover disabled:bg-opacity-80':
```

```
            variant === 'outlined',
          'text-md  h-[42px] min-h-[42px] w-[42px] min-w-[42px] rounded-2xl
bg-primary enabled:hover:brightness-hover disabled:brightness-50':
            variant === 'circle',
          'rounded-xl p-1 text-lg text-primary hover:brightness-hover':
            variant === 'text',
          'rounded-xl p-2 text-lg text-primary': variant === 'action',
        },
      )}
    >
      {getLinkWrapper(
        <>
          {loading && <Loader />}
          {children}
        </>,
      )}
    </button>
  );
}
```

Фрагменти коду шаблонів

```
export type AuthTemplateProps = {
  children: React.ReactNode;
};

export function AuthTemplate({ children }: AuthTemplateProps) {
  return (
    <div className="flex min-h-screen w-full justify-center p-5">
      <div className="flex max-w-[500px] flex-1 flex-col justify-center
gap-5">
        <h1 className="mb-5 text-center text-7xl font-semibold">Smartner</h1>

        {children}
      </div>
    </div>
  );
}
```

Фрагменти коду сторінок

```
import { useRouter } from 'next/router';

import { Separator } from '@/components/atoms';
import { Button, InputText } from '@/components/molecules';
import { AuthTemplate } from '@/components/templates';
import { useClientForm, useSignIn } from '@/libs/hooks';
import { SignInSchema } from '@/libs/schemas';

export default function SignIn() {
  const router = useRouter();
  const form = useClientForm({ schema: SignInSchema });
  const signIn = useSignIn();

  const handleSubmit = async (values: SignInSchema) => {
    await signIn.mutateAsync(values);
    void router.push({ pathname: '/board' });
  };

  return (
    <AuthTemplate>
```

```
      <form
        className="flex flex-col gap-5"
        onSubmit={form.handleSubmit(handleSubmit)}
      >
        <InputText
          control={form.control}
          name="email"
          label="Email"
          placeholder="Enter email"
          type="email"
        />
        <InputText
          control={form.control}
          name="password"
          label="Password"
          placeholder="Enter password"
          type="password"
        />

        <Button type="submit" className="mt-2">
          Sign in
        </Button>
      </form>

      <Separator />

      <Button to={{ pathname: '/sign-up' }} variant="text">
        Sign up new account
      </Button>
    </AuthTemplate>
  );
}

import {
  InboxIcon,
  PlusCircleIcon,
  RefreshCcwIcon,
  SortDescIcon,
} from 'lucide-react';
import { z } from 'zod';

import { Separator } from '@/components/atoms';
import { ButtonIcon, InputText } from '@/components/molecules';
import { BoardModal } from '@/components/organisms/board-modal';
import { BoardPaper } from '@/components/organisms/board-paper';
import { PageTemplate } from '@/components/templates/page-template';
import { OrderType } from '@/libs/enums/order-type';
import { useClientForm } from '@/libs/hooks';
import { useModal } from '@/libs/hooks';
import { useBoards } from '@/queries/board.queries';
import { useStore } from '@/store';

const schema = z.object({
  search: z.string(),
});

export default function Boards() {
  const boards = useBoards();
  const { boardFilters, setBoardFilters } = useStore();
  const modal = useModal();
  const form = useClientForm({ schema, defaultValues: { search: '' } });

  const handleSearch = (search?: string) => {
    setBoardFilters({ search: search ?? '' });
```

```
  };

  const handleSort = () => {
    setBoardFilters({
      order:
        boardFilters.order === OrderType.DESC ? OrderType.ASC :
OrderType.DESC,
    });
  };

  const handleRefresh = () => {
    void boards.refetch();
  };

  return (
    <>
      <PageTemplate
        title="Boards"
        loading={boards.isLoading}
        header={
          <>
            <InputText
              control={form.control}
              name="search"
              placeholder="Search"
              className="mr-2"
              hideDetails
              onChange={handleSearch}
            />

            <ButtonIcon icon={<SortDescIcon />} onClick={handleSort} />
            <ButtonIcon icon={<RefreshCcwIcon />} onClick={handleRefresh} />

            <Separator vertical />

            <ButtonIcon icon={<PlusCircleIcon />} onClick={modal.open} />
            <ButtonIcon
              icon={<InboxIcon />}
              to={{ pathname: '/board/invite' }}
            />
          </>
        }
      >
        {boards.data?.map((item) => <BoardPaper {...item} />)}
      </PageTemplate>
      <BoardModal isOpen={modal.isOpen} onClose={modal.close} />
    </>
  );
}

import { EyeIcon, RefreshCcwIcon, Trash2Icon } from 'lucide-react';

import { Separator } from '@/components/atoms';
import { ButtonIcon } from '@/components/molecules';
import { NotificationPaper } from
'@/components/organisms/notification-paper';
import { PageTemplate } from '@/components/templates/page-template';
import {
  useClearNotifications,
  useNotifications,
  useReadNotifications,
} from '@/queries/notification.queries';

export default function Notifications() {
```

```
  const { data, isLoading } = useNotifications();
  const read = useReadNotifications();
  const clear = useClearNotifications();

  return (
    <PageTemplate
      title="Notifications"
      loading={isLoading}
      header={
        <>
          <div className="flex-1" />

          <ButtonIcon icon={<RefreshCcwIcon />} />

          <Separator vertical />

          <ButtonIcon
            icon={<EyeIcon />}
            loading={read.isLoading}
            onClick={() => read.mutateAsync()}
          />
          <ButtonIcon
            icon={<Trash2Icon />}
            className="bg-error"
            loading={clear.isLoading}
            onClick={() => clear.mutateAsync()}
          />
        </>
      }
    >
      {data?.map((item) => <NotificationPaper {...item} />)}
      {!data?.length && (
        <span className="text-center">You do not have new
notifications</span>
      )}
    </PageTemplate>
  );
}
```

Фрагменти модуля користувача

```
package user

import (
  "time"

  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/types"
)

type FindOneQueryParams struct {
  ID    types.Optional[types.ID]
  Email types.Optional[string]
}

type FindOneCredsQueryParams struct {
  Email    string
  Nickname string
}

type FindPageQueryParams struct {
  IDs    []types.ID
  NotIDs []types.ID
}
```

```go
type CreateQueryParams struct {
  FirstName string
  LastName  string
  Nickname  string
  Email     string
  AvatarURL types.Optional[string]
}

type UpdateQueryParams struct {
  ID        types.ID
  FirstName types.Optional[string]
  LastName  types.Optional[string]
  AvatarKey types.Optional[string]
}

type QueryItem struct {
  ID        types.ID
  FirstName string
  LastName  string
  Nickname  string
  Email     string
  AvatarKey *string
  CreatedAt time.Time
  UpdatedAt time.Time
}

type GetOneParams struct {
  ID    *types.ID
  Email *string
}

type GetPageParams = FindPageQueryParams

type CreateParams struct {
  FirstName string
  LastName  string
  Nickname  string
  Email     string
  Avatar    *types.File
}

type UpdateParams struct {
  FirstName   *string
  LastName    *string
  Avatar      *types.File
  ClearAvatar bool
}

type Item struct {
  QueryItem
  AvatarURL *string
}

package user

import (
  "context"

  models "github.com/dmytro-kucherenko/smartner-server/internal/models/user"
  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/types"
)

type Repo interface {
```

```
  Count(context.Context, models.FindPageQueryParams) (total uint64, err
error)
  FindOne(context.Context, models.FindOneQueryParams) (models.QueryItem,
error)
  FindOneCreds(context.Context, models.FindOneCredsQueryParams)
(models.QueryItem, error)
  FindPage(context.Context, models.FindPageQueryParams) (items
[]models.QueryItem, err error)
  Create(context.Context, models.CreateQueryParams) (models.QueryItem, error)
  Update(context.Context, models.UpdateQueryParams) (models.QueryItem, error)
}

type FileRepo interface {
  BuildAvatarKey(*types.File, string) string
  GenerateLink(context.Context, string) (types.FileLink, error)
  GenerateManyLink(context.Context, ...string) ([]types.FileLink, error)
  Upload(context.Context, *types.File) error
  DeleteMany(context.Context, ...string) error
}


package user

import (
  "context"

  models "github.com/dmytro-kucherenko/smartner-server/internal/models/user"
  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/meta"
  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/types"
)

type Service struct {
  repo     Repo
  fileRepo FileRepo
}

func NewService(repository Repo, fileRepo FileRepo) *Service {
  return &Service{repository, fileRepo}
}

func (s *Service) GetOne(ctx context.Context, params models.GetOneParams)
(item models.Item, err error) {
  queryItem, err := s.repo.FindOne(ctx, models.FindOneQueryParams{
    ID:    types.OptionalPointer(params.ID),
    Email: types.OptionalPointer(params.Email),
  })
  if err != nil {
    err = NewErrNotFound()
    return
  }

  item.QueryItem = queryItem
  if queryItem.AvatarKey == nil {
    return
  }

  avatarLink, err := s.fileRepo.GenerateLink(ctx, *queryItem.AvatarKey)
  if err != nil {
    return
  }

  item.AvatarURL = &avatarLink.URL
  return
}
```

```
func (s *Service) GetPage(ctx context.Context, params models.GetPageParams)
(items []models.Item, err error) {
  queryItems, err := s.repo.FindPage(ctx, models.FindPageQueryParams{IDs:
params.IDs})
  if err != nil {
    return
  }

  keys := make([]string, 0, len(queryItems))
  for _, item := range queryItems {
    if item.AvatarKey != nil {
      keys = append(keys, *item.AvatarKey)
    }
  }

  items = mapItems(queryItems, []types.FileLink{})
  if len(keys) == 0 {
    return
  }

  links, err := s.fileRepo.GenerateManyLink(ctx, keys...)
  if err != nil {
    return
  }

  items = mapItems(queryItems, links)
  return
}

func (s *Service) Create(ctx context.Context, params models.CreateParams)
(item models.QueryItem, err error) {
  _, err = s.repo.FindOneCreds(ctx, models.FindOneCredsQueryParams{
    Email:    params.Email,
    Nickname: params.Nickname,
  })
  if err == nil {
    err = NewErrCredsExist()
    return
  }

  avatarKey := s.fileRepo.BuildAvatarKey(params.Avatar, params.Nickname)
  item, err = s.repo.Create(ctx, models.CreateQueryParams{
    FirstName: params.FirstName,
    LastName:  params.LastName,
    Nickname:  params.Nickname,
    Email:     params.Email,
    AvatarURL: types.OptionalZeroed(avatarKey),
  })
  if err != nil || params.Avatar == nil {
    return
  }

  s.fileRepo.Upload(ctx, params.Avatar)

  return
}

func (s *Service) Update(ctx context.Context, params models.UpdateParams)
(item models.QueryItem, err error) {
  session, _ := meta.GetSession(ctx)

  avatarKey := types.OptionalZeroed(s.fileRepo.BuildAvatarKey(params.Avatar,
session.User.Nickname))
```

```
  if !avatarKey.Valid && params.ClearAvatar {
    avatarKey.Valid = true
  }

  item, err = s.repo.Update(ctx, models.UpdateQueryParams{
    ID:        session.User.ID,
    FirstName: types.OptionalPointer(params.FirstName),
    LastName:  types.OptionalPointer(params.LastName),
    AvatarKey: avatarKey,
  })
  if err != nil || !avatarKey.Valid {
    return
  }

  if avatarKey.Value != "" {
    s.fileRepo.Upload(ctx, params.Avatar)
  }

  if item.AvatarKey != nil {
    s.fileRepo.DeleteMany(ctx, *item.AvatarKey)
  }

  return
}

package user

import (
  "context"
  "database/sql"

  "github.com/google/uuid"

  db "github.com/dmytro-kucherenko/smartner-server/internal/gen/db/user"
  models "github.com/dmytro-kucherenko/smartner-server/internal/models/user"
  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/queries"
)

type Repo struct {
  manager queries.Manager[*db.Queries]
  queries.TransactionCaller[*db.Queries]
}

func NewRepo(connection *sql.DB) *Repo {
  manager := queries.NewSQLManager(
    connection,
    func(connection *sql.DB) *db.Queries { return db.New(connection) },
    func(ctx context.Context, connection *sql.DB) (*db.Queries, error) {
return db.Prepare(ctx, connection) },
  )

  return &Repo{
    manager:           manager,
    TransactionCaller: manager,
  }
}

func (r *Repo) queries() *db.Queries {
  return r.manager.Queries()
}

func (r *Repo) Count(ctx context.Context, params models.FindPageQueryParams)
(total uint64, err error) {
  count, err := r.queries().Count(ctx, params.IDs)
```

```
  if err != nil {
    return
  }

  return uint64(count), nil
}

func (r *Repo) FindOne(ctx context.Context, params models.FindOneQueryParams)
(models.QueryItem, error) {
  user, err := r.queries().FindOne(ctx, uuid.NullUUID{UUID: params.ID.Value,
Valid: params.ID.Valid})

  return mapItem(user), err
}

func (r *Repo) FindOneCreds(ctx context.Context, params
models.FindOneCredsQueryParams) (models.QueryItem, error) {
  user, err := r.queries().FindOneCreds(ctx, db.FindOneCredsParams{
    Nickname: params.Nickname,
    Email:    params.Email,
  })

  return mapItem(user), err
}

func (r *Repo) FindPage(ctx context.Context, params
models.FindPageQueryParams) (items []models.QueryItem, err error) {
  users, err := r.queries().FindPage(ctx, db.FindPageParams{
    Ids:    params.IDs,
    NotIds: params.NotIDs,
  })
  if err != nil {
    return
  }

  return mapItems(users), nil
}

func (r *Repo) Create(ctx context.Context, params models.CreateQueryParams)
(models.QueryItem, error) {
  user, err := r.queries().Create(ctx, db.CreateParams{
    FirstName: params.FirstName,
    LastName:  params.LastName,
    Nickname:  params.Nickname,
    Email:     params.Email,
    AvatarKey: sql.NullString{String: params.AvatarURL.Value, Valid:
params.AvatarURL.Valid},
  })

  return mapItem(user), err
}

func (r *Repo) Update(ctx context.Context, params models.UpdateQueryParams)
(models.QueryItem, error) {
  user, err := r.queries().Update(ctx, db.UpdateParams{
    ID:        params.ID,
    FirstName: sql.NullString{String: params.FirstName.Value, Valid:
params.FirstName.Valid},
    LastName:  sql.NullString{String: params.LastName.Value, Valid:
params.LastName.Valid},
    AvatarKey: sql.NullString{String: params.AvatarKey.Value, Valid:
params.AvatarKey.Valid},
  })
```

```
  return mapItem(user), err
}

package user

import (
  "fmt"

  "github.com/aws/aws-sdk-go-v2/service/s3"
  s3Infra "github.com/dmytro-kucherenko/smartner-server/internal/infra/s3"
  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/types"
)

type S3Repo struct {
  s3Infra.Repo
}

func NewS3Repo(client *s3.Client, bucket string) *S3Repo {
  s3Repo := s3Infra.NewRepo(client, bucket, "user")

  return &S3Repo{*s3Repo}
}

func (r *S3Repo) BuildAvatarKey(data *types.File, nickname string) string {
  if data == nil {
    return ""
  }

  key := r.BuildKey(fmt.Sprintf("%s_avatar", nickname))
  data.Key = key

  return key
}

-- name: Count :one
SELECT
    COUNT(*)
FROM
    users
WHERE
    id IN (sqlc.narg('ids')::uuid[]);

-- name: FindOne :one
SELECT *
FROM
    users
WHERE
    sqlc.narg('id')::uuid IS NULL
    OR id = sqlc.narg('id')::uuid
LIMIT 1;

-- name: FindOneCreds :one
SELECT *
FROM
    users
WHERE
    nickname = sqlc.arg('nickname')::varchar
    OR email = sqlc.arg('email')::varchar
LIMIT 1;

-- name: FindPage :many
SELECT *
FROM
    users
```

```
WHERE
    (sqlc.narg('ids')::uuid[] IS NULL OR id IN (sqlc.narg('ids')::uuid[]))
    AND (sqlc.narg('not_ids')::uuid[] IS NULL OR id NOT IN
(sqlc.narg('not_ids')::uuid[]));

-- name: Create :one
INSERT INTO users (
    first_name,
    last_name,
    nickname,
    email,
    avatar_key
)
VALUES (
    sqlc.arg('first_name'),
    sqlc.arg('last_name'),
    sqlc.arg('nickname'),
    sqlc.arg('email'),
    sqlc.arg('avatar_key')
)
RETURNING *;

-- name: Update :one
UPDATE users
SET
    first_name = COALESCE(sqlc.narg('first_name')::varchar, first_name),
    last_name  = COALESCE(sqlc.narg('last_name')::varchar, last_name),
    avatar_key = COALESCE(sqlc.narg('avatar_key')::varchar, avatar_key)
WHERE
    id = sqlc.arg(id)
RETURNING *;
```

```
package user

import (
  "context"
  "net/http"

  adapter
"github.com/dmytro-kucherenko/smartner-server/internal/pkg/server/adapters/gi
n"
  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/utils"
  "github.com/gin-gonic/gin"
)

type Handler struct {
  authInt gin.HandlerFunc
  service Service
}

func NewHandler(authInt gin.HandlerFunc, service Service) *Handler {
  return &Handler{authInt, service}
}

func (h *Handler) Init(group *gin.RouterGroup) {
  userGroup := group.Group("/user")
  config := adapter.NewConfig().WithInterceptor(h.authInt)
  adapter.Put(userGroup, h.update, config.MapRoute("/update", http.StatusOK))
}

// @Summary  Update profile
// @Tags          User
// @Security JWTAuth
// @Accept        multipart/form-data
```

```go
// @Produce  json
// @Param         body  body        UserUpdateRequest  true   "User Data"
// @Success  200         {object}    UserResponse
// @Failure  400         {object}    ErrorResponse
// @Failure  401         {object}    ErrorResponse
// @Failure  404         {object}    ErrorResponse
// @Router        /api/user/update [put]
func (h *Handler) update(ctx context.Context, params UpdateRequest)
(Response, error) {
  return utils.MapHandler(h.service.Update, mapUpdateParams, MapQueryItem,
mapError)(ctx, params)
}

package user

import (
  "net/http"

  domain "github.com/dmytro-kucherenko/smartner-server/internal/domain/user"
  models "github.com/dmytro-kucherenko/smartner-server/internal/models/user"
  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/errors"
  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/meta"
  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/types"
)

func mapUpdateParams(params UpdateRequest) (result models.UpdateParams, err
error) {
  avatar, err := types.ParseMultipartFile(params.Avatar)
  if err != nil {
    return
  }

  return models.UpdateParams{
    FirstName:   params.FirstName,
    LastName:    params.LastName,
    Avatar:      avatar,
    ClearAvatar: params.ClearAvatar,
  }, nil
}

func MapQueryItem(item models.QueryItem) Response {
  return Response{
    ID:        item.ID,
    FirstName: item.FirstName,
    LastName:  item.LastName,
    Nickname:  item.Nickname,
    Email:     item.Email,
    CreatedAt: item.CreatedAt,
    UpdatedAt: item.UpdatedAt,
  }
}

func MapItem(item models.Item) Response {
  response := MapQueryItem(item.QueryItem)
  response.AvatarURL = item.AvatarURL
  return response
}

func MapSession(session meta.SessionUser) Response {
  return Response{
    ID:        session.ID,
    FirstName: session.FirstName,
    LastName:  session.LastName,
    Nickname:  session.Nickname,
```

```
    Email:     session.Email,
    AvatarURL: session.AvatarURL,
    CreatedAt: session.CreatedAt,
    UpdatedAt: session.UpdatedAt,
  }
}

func MapItems(items []models.Item) []Response {
  users := make([]Response, 0, len(items))
  for _, item := range items {
    users = append(users, MapItem(item))
  }

  return users
}

func mapError(err error) error {
  if !domain.IsError(err) {
    return err
  }

  status := http.StatusInternalServerError
  switch code, _ := errors.AppCode(err); code {
  case domain.ErrNotFound:
    status = http.StatusNotFound
  case domain.ErrCredsExist:
    status = http.StatusBadRequest
  }

  return errors.NewHTTPFromError(status, err)
}
```

Фрагменти модуля збереження файлів

```
package s3

import (
  "bytes"
  "context"
  "fmt"
  "sync"
  "time"

  "github.com/aws/aws-sdk-go-v2/aws"
  "github.com/aws/aws-sdk-go-v2/service/s3"
  "github.com/aws/aws-sdk-go-v2/service/s3/types"
  appErrors
"github.com/dmytro-kucherenko/smartner-server/internal/pkg/errors"
  types2 "github.com/dmytro-kucherenko/smartner-server/internal/pkg/types"
  "github.com/google/uuid"
)

type Repo struct {
  client        *s3.Client
  presignClient *s3.PresignClient
  bucket        string
  directory     string
}

func NewRepoWithExpiration(client *s3.Client, bucket string, directory
string, presignedExpiration time.Duration) *Repo {
```

```
  presignClient := s3.NewPresignClient(client, func(opts *s3.PresignOptions)
{
    opts.Expires = presignedExpiration
  })

  return &Repo{client, presignClient, bucket, directory}
}

func NewRepo(client *s3.Client, bucket string, directory string) *Repo {
  return NewRepoWithExpiration(client, bucket, directory, time.Hour)
}

func (s *Repo) BuildKey(key string) string {
  if s.directory == "" {
    return key
  }

  return fmt.Sprintf("%s/%s_%s_%s", s.directory, key, uuid.New(),
time.Now().Format(time.RFC3339)) // what if uuid panic
}

func (s *Repo) GenerateLink(ctx context.Context, key string) (link
types2.FileLink, err error) {
  request, err := s.presignClient.PresignGetObject(ctx, &s3.GetObjectInput{
    Bucket: aws.String(s.bucket),
    Key:    aws.String(key),
  })

  if err != nil {
    err = mapError(err, "failed to generate file URL")
    return
  }

  return types2.FileLink{
    Key: key,
    URL: request.URL,
  }, nil
}

func (s *Repo) GenerateManyLink(ctx context.Context, keys ...string) (links
[]types2.FileLink, err error) {
  var wg sync.WaitGroup
  responses := make(chan types2.FileLink, len(keys))
  errs := make(chan error, 1)
  ctx, cancel := context.WithCancel(ctx)
  defer cancel()

  for _, key := range keys {
    wg.Add(1)
    go func() {
      defer wg.Done()

      response, err := s.GenerateLink(ctx, key)
      if err != nil {
        errs <- err
        return
      }

      responses <- response
    }()
  }

  go func() {
    wg.Wait()
```

```
    close(responses)
    close(errs)
  }()

  links = make([]types2.FileLink, 0, len(keys))
  for {
    select {
    case link, ok := <-responses:
      if ok {
        links = append(links, link)
      }
    case err, ok := <-errs:
      if ok {
        cancel()
        return links, err
      }
    }
  }
}

func (s *Repo) Upload(ctx context.Context, data *types2.File) error {
  if data == nil {
    return appErrors.NewApp(appErrors.CodeValidation, "file data required")
  }

  _, err := s.client.PutObject(ctx, &s3.PutObjectInput{
    Bucket:      aws.String(s.bucket),
    Key:         aws.String(data.Key),
    Body:        bytes.NewReader(data.Content),
    ContentType: aws.String(data.ContentType),
    ACL:         types.ObjectCannedACLPrivate,
  })
  if err != nil {
    return appErrors.NewApp(appErrors.CodeInternal, "failed to upload
file").FatalMessage(err.Error())
  }

  return nil
}

func (s *Repo) UploadMany(ctx context.Context, data ...*types2.File) error {
  var wg sync.WaitGroup
  errs := make(chan error, 1)
  cancelCtx, cancel := context.WithCancel(ctx)
  defer cancel()

  for _, fileData := range data {
    wg.Add(1)
    go func() {
      defer wg.Done()

      err := s.Upload(cancelCtx, fileData)
      if err != nil {
        errs <- err
        return
      }
    }()
  }

  go func() {
    wg.Wait()
    close(errs)
  }()
```

```
  select {
  case err, ok := <-errs:
    if ok {
      cancel()
      keys := make([]string, 0, len(data))
      for _, data := range data {
        keys = append(keys, data.Key)
      }

      _ = s.DeleteMany(ctx, keys...)
      return err
    }
  }

  return nil
}

func (s *Repo) DeleteMany(ctx context.Context, keys ...string) error {
  if len(keys) == 0 {
    return nil
  }

  objects := make([]types.ObjectIdentifier, 0, len(keys))
  for _, key := range keys {
    objects = append(objects, types.ObjectIdentifier{
      Key: aws.String(key),
    })
  }

  _, err := s.client.DeleteObjects(ctx, &s3.DeleteObjectsInput{
    Bucket: aws.String(s.bucket),
    Delete: &types.Delete{Objects: objects},
  })

  return mapError(err, "failed to delete files")
}

package s3

import (
  "errors"

  "github.com/aws/aws-sdk-go-v2/service/s3/types"
  appErrors
"github.com/dmytro-kucherenko/smartner-server/internal/pkg/errors"
)

func mapError(err error, message string) error {
  if err == nil {
    return nil
  }

  if noKey := new(types.NoSuchKey); errors.As(err, &noKey) {
    return appErrors.NewApp(appErrors.CodeFileNotFound, "file not found")
  }

  return appErrors.NewApp(appErrors.CodeInternal,
message).FatalMessage(err.Error())
}
```

Фрагменти модуля токенізації даних

```
package jwt
```

```
import (
  "time"

  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/errors"
  "github.com/golang-jwt/jwt/v5"
)

type claims[T any] struct {
  jwt.RegisteredClaims
  Data T `json:"data"`
}

type Service[T any] struct {
  secret     []byte
  expiration time.Duration
  method     jwt.SigningMethod
}

func NewService[T any](secret string, expiration time.Duration) *Service[T] {
  return &Service[T]{[]byte(secret), expiration, jwt.SigningMethodHS256}
}

func (s *Service[T]) key(token *jwt.Token) (interface{}, error) {
  if token.Method.Alg() == s.method.Alg() {
    return nil, errors.NewApp(errors.CodeValidation, "signing method
invalid")
  }

  return s.secret, nil
}

func (s *Service[T]) Gen(data T) (token string, expiresAt time.Time, err
error) {
  now := time.Now()
  expiresAt = now.Add(s.expiration)
  tokenClaims := claims[T]{
    Data: data,
    RegisteredClaims: jwt.RegisteredClaims{
      Issuer:    "smartner",
      IssuedAt:  jwt.NewNumericDate(now),
      ExpiresAt: jwt.NewNumericDate(expiresAt),
    },
  }

  token, err = jwt.NewWithClaims(s.method,
tokenClaims).SignedString(s.secret)
  if err != nil {
    err = errors.NewApp(errors.CodeInternal, "internal
error").FatalMessage(err.Error())
    return
  }

  return
}

func (s *Service[T]) Verify(token string) (data T, expired bool, err error) {
  var tokenClaims claims[T]
  parsedToken, err := jwt.ParseWithClaims(token, &tokenClaims, s.key)
  if err != nil || !parsedToken.Valid {
    err = errors.NewApp(errors.CodeValidation, "token invalid")
    return
  }
```

```
  data = tokenClaims.Data
  expired = tokenClaims.ExpiresAt.Before(time.Now())

  return
}
```

Фрагменти модуля шифрування даних

```
package encrypt

import (
  "crypto/rand"
  "crypto/sha256"
  "crypto/subtle"
  "encoding/hex"
  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/errors"
)

type Service struct {
  secret string
  rounds uint8
}

func NewService(secret string, rounds uint8) *Service {
  return &Service{secret, rounds}
}

func (service *Service) genSalt() (salt string, err error) {
  sequence := make([]byte, service.rounds)
  _, err = rand.Read(sequence)
  if err != nil {
    err = errors.NewApp(errors.CodeInternal, "Failed to encrypt password.")

    return
  }

  salt = hex.EncodeToString(sequence)

  return
}

func (service *Service) get(data string, salt string) string {
  hasher := sha256.New()
  hasher.Write([]byte(data + salt + service.secret))

  hash := hex.EncodeToString(hasher.Sum(nil))

  return hash
}

func (service *Service) Gen(data string) (hash, salt string, err error) {
  salt, err = service.genSalt()
  if err != nil {
    return
  }

  hash = service.get(data, salt)

  return
}

func (service *Service) Verify(data string, hash, salt string) bool {
  computedHash := service.get(data, salt)
```

```
  return subtle.ConstantTimeCompare([]byte(computedHash), []byte(hash)) == 1
}
```

Фрагменти ініціалізації серверної частини

```
package internal

import (
  "context"
  "fmt"
  "net/http"

  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/queries"
  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/schema"
  adapter
"github.com/dmytro-kucherenko/smartner-server/internal/pkg/server/adapters/gi
n"
  "github.com/gin-gonic/gin"

  awsConfig "github.com/aws/aws-sdk-go-v2/config"
  "github.com/aws/aws-sdk-go-v2/service/s3"
  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/app"
  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/config"
  "github.com/dmytro-kucherenko/smartner-server/internal/pkg/log"
  restInterceptors
"github.com/dmytro-kucherenko/smartner-server/internal/pkg/server/adapters/gi
n/interceptors"

"github.com/dmytro-kucherenko/smartner-server/internal/pkg/server/multiplexer
"
  _ "github.com/lib/pq"
  swaggerfiles "github.com/swaggo/files"
  ginSwagger "github.com/swaggo/gin-swagger"
)

func newApp() (instance *app.Instance, err error) {
  err = config.Init()
  if err != nil {
    return
  }

  err = schema.Init()
  if err != nil {
    return
  }

  connection := config.DBConnection()
  db, err := queries.ConnectSQL(connection)
  if err != nil {
    return
  }

  awsCreds, err := awsConfig.LoadDefaultConfig(context.Background())
  if err != nil {
    return
  }
  s3Client := s3.NewFromConfig(awsCreds)

  builder := func() []adapter.GinHandler {
    return build(db, s3Client)
  }
```

```go
  return app.New(builder).Build(), nil
}

func initREST(instance *app.Instance) *http.Server {
  logger := log.New("API")
  clientURL := config.Schema.ClientURL
  isProd := config.IsProd()

  if isProd {
    gin.SetMode(gin.ReleaseMode)
  }

  router := gin.New()
  httpServer := &http.Server{Handler: router.Handler()}
  if isProd {
    router.SetTrustedProxies([]string{clientURL})
  }

  api := router.Group(fmt.Sprint(config.Schema.AppBasePath, "/api"))
  api.Use(
    restInterceptors.Logger(logger),
    restInterceptors.Error(),
    restInterceptors.Options(),
  )
  api.GET("/swagger/*any", ginSwagger.WrapHandler(swaggerfiles.Handler))

  instance.Init(api)

  return httpServer
}

func serve(logger log.Logger) error {
  application, err := newApp()
  if err != nil {
    return err
  }

  port := config.AppPort()
  instance, err := multiplexer.New(port)
  if err != nil {
    return err
  }

  return instance.
    WithLogger(logger).
    WithREST(initREST(application)).
    ServeGracefully()
}

func Run() {
  logger := log.New("Server")
  if err := serve(logger); err != nil {
    logger.Fatal(err.Error())
  }
}
```

Фрагменти міграцій бази даних

```sql
CREATE TYPE board_member_role AS enum ('landlord', 'tenant');

CREATE TABLE boards (
    id          uuid PRIMARY KEY      DEFAULT uuid_generate_v4(),
```

```
    owner_id    uuid         NOT NULL REFERENCES users (id) ON DELETE
CASCADE,
    name        varchar(255) NOT NULL,
    description text,
    avatar_key  text,
    created_at  timestamp    NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at  timestamp    NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TRIGGER update_boards_columns
    BEFORE
        UPDATE
    ON boards
    FOR EACH ROW
EXECUTE FUNCTION update_columns();

CREATE TABLE board_members (
    id         uuid PRIMARY KEY           DEFAULT uuid_generate_v4(),
    board_id   uuid              NOT NULL REFERENCES boards (id) ON DELETE
CASCADE,
    member_id  uuid              NOT NULL REFERENCES users (id) ON DELETE
CASCADE,
    role       board_member_role NOT NULL,
    is_invite  boolean           NOT NULL DEFAULT TRUE,
    created_at timestamp         NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at timestamp         NOT NULL DEFAULT CURRENT_TIMESTAMP,
    UNIQUE (board_id, member_id)
);

CREATE TRIGGER update_board_members_columns
    BEFORE
        UPDATE
    ON board_members
    FOR EACH ROW
EXECUTE FUNCTION update_columns();

CREATE TABLE board_items (
    id          uuid PRIMARY KEY      DEFAULT uuid_generate_v4(),
    board_id    uuid         NOT NULL REFERENCES boards (id) ON DELETE
CASCADE,
    owner_id    uuid         REFERENCES board_members (id) ON DELETE SET
NULL,
    owner_role  board_member_role,
    name        varchar(255) NOT NULL UNIQUE,
    description text,
    created_at  timestamp    NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at  timestamp    NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TRIGGER update_board_items_columns
    BEFORE
        UPDATE
    ON board_items
    FOR EACH ROW
EXECUTE FUNCTION update_columns();

CREATE TABLE board_item_images (
    id         uuid PRIMARY KEY   DEFAULT uuid_generate_v4(),
    item_id    uuid      NOT NULL REFERENCES board_items (id) ON DELETE
CASCADE,
    key        text      NOT NULL,
    created_at timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

```
CREATE TRIGGER update_board_item_images_columns
    BEFORE
        UPDATE
    ON board_item_images
    FOR EACH ROW
EXECUTE FUNCTION update_columns();

CREATE TYPE board_change_type AS enum (
    'name_updated',
    'description_updated',
    'avatar_updated',
    'member_invited',
    'member_accepted_invite',
    'member_declined_invite',
    'item_created',
    'item_deleted',
    'item_owner_role_updated',
    'item_owner_id_updated',
    'item_name_updated',
    'item_description_updated',
    'item_image_created',
    'item_image_deleted'
    );

CREATE TABLE board_changes (
    id         uuid PRIMARY KEY           DEFAULT uuid_generate_v4(),
    board_id   uuid              NOT NULL REFERENCES boards (id) ON DELETE
CASCADE,
    author_id  uuid REFERENCES users (id) NOT NULL,
    member_id  uuid REFERENCES users (id),
    item_name  varchar(255),
    type       board_change_type NOT NULL,
    value      text,
    created_at timestamp         NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

Додаток 3

Копія презента

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ФІКСУВАННЯ МАЙНА ОРЕНДАРЯ ТА ОРЕНДОДАВЦЯ ПІД ЧАС НАДАННЯ ПОСЛУГ З ОРЕНДИ

Виконав: Дмитро Кучеренко, КП-11

Керівник: Асистент кафедри ПЗКС, д-р філософії Яків Юсин

Київ – 2025

ПОСТАНОВКА ЗАДАЧІ

Мета проєкту: покращити якість документування стану майна під час процесу орендування.

Завдання:

1. Дослідження особливостей процесу та існуючих програмних рішень.
2. Визначення основних вимог до системи.
3. Вибір та обґрунтування оптимальних засобів реалізації застосунку.
4. Розроблення компонентів системи.
5. Тестування розробленого застосунку.
6. Порівняння з аналогами.
7. Опис напрямків подальшого розвитку проєкту.

АКТУАЛЬНІСТЬ

- Відсутність зручних цифрових інструментів для обліку стану майна при укладанні договорів оренди.
- Необхідність покращення прозорості відносин між орендодавцями та орендарями щодо стану майна.
- Зростаючий попит на автоматизовані рішення для документування змін майна протягом терміну використання.
- Можливість для масштабування рішення для агентств нерухомості та приватних власників.

НАЯВНІ АНАЛОГИ

Застосунок «MRI Property Tree Connect»

Застосунок «Rental Property Management»

Паперові договори

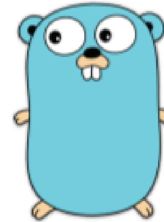
ФУНКЦІОНАЛЬНІ ВИМОГИ ДО РОЗРОБЛЮВАНОГО ПЗ

Вебзастосунок повинен забезпечувати такі основні вимоги:

1. Створення облікового запису та вхід до нього, редагування особистих даних.
2. Створення нової дошки приміщення та редагування її майна.
3. Перегляд історії змін дошки.
4. Запрошення та видалення учасників дошки.
5. Вихід з дошки або її видалення залежно від ролі користувача.
6. Перегляд, прочитання та видалення сповіщень.
7. Налаштування отримання сповіщень.

ОБРАНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Next.js



Golang

TailwindCSS

PostgreSQL

Схема архітектури серверної частини

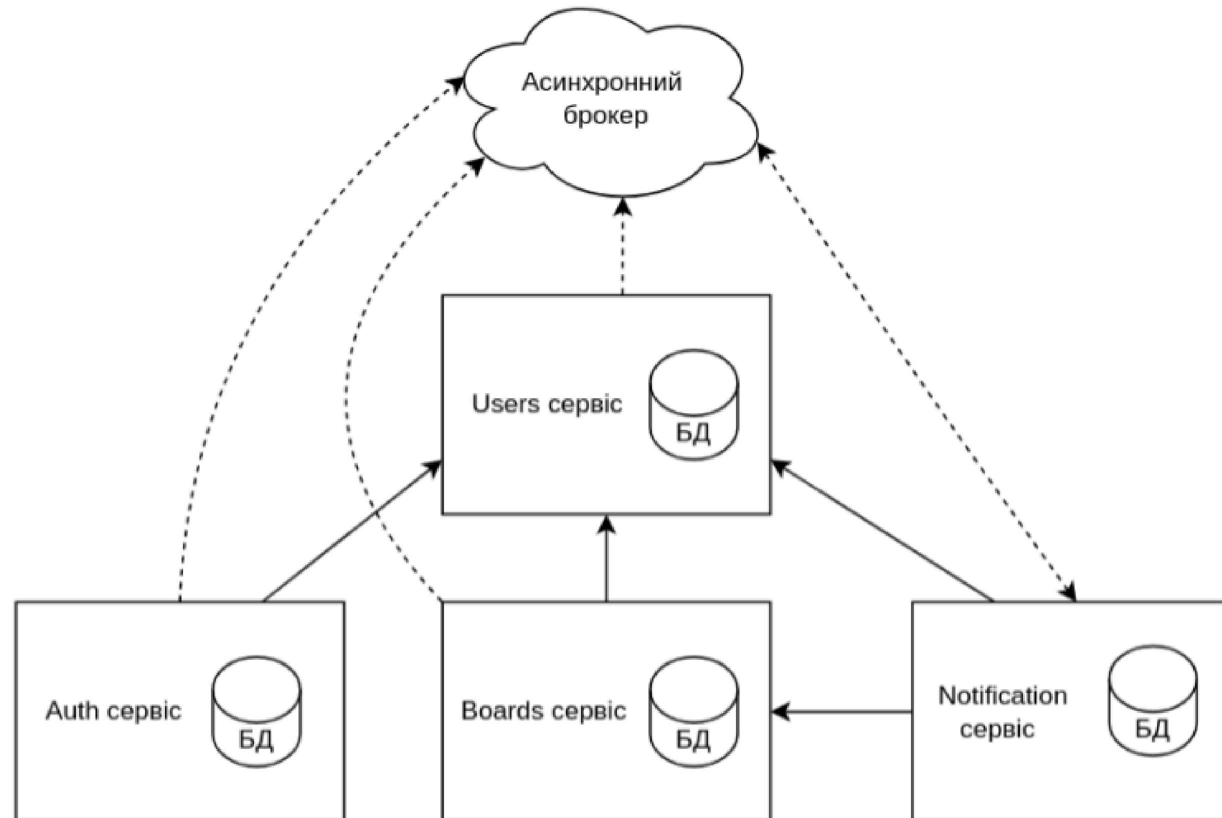


СХЕМА АРХІТЕКТУРИ КОЖНОГО МІКРОСЕРВІСУ

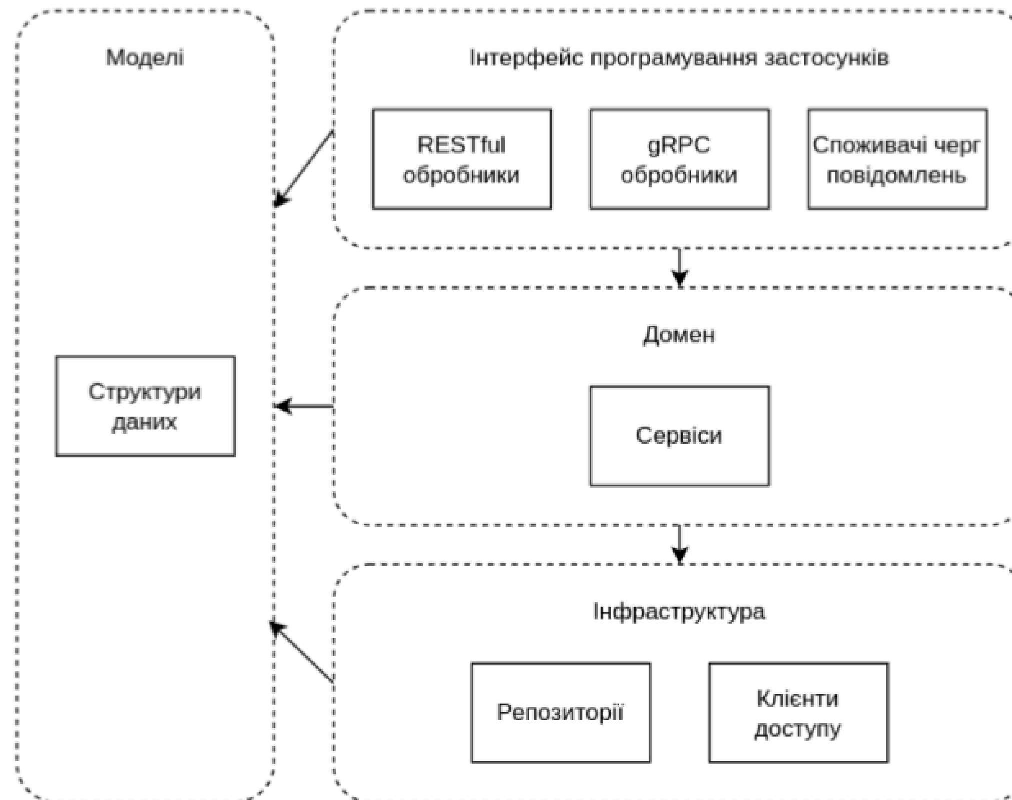
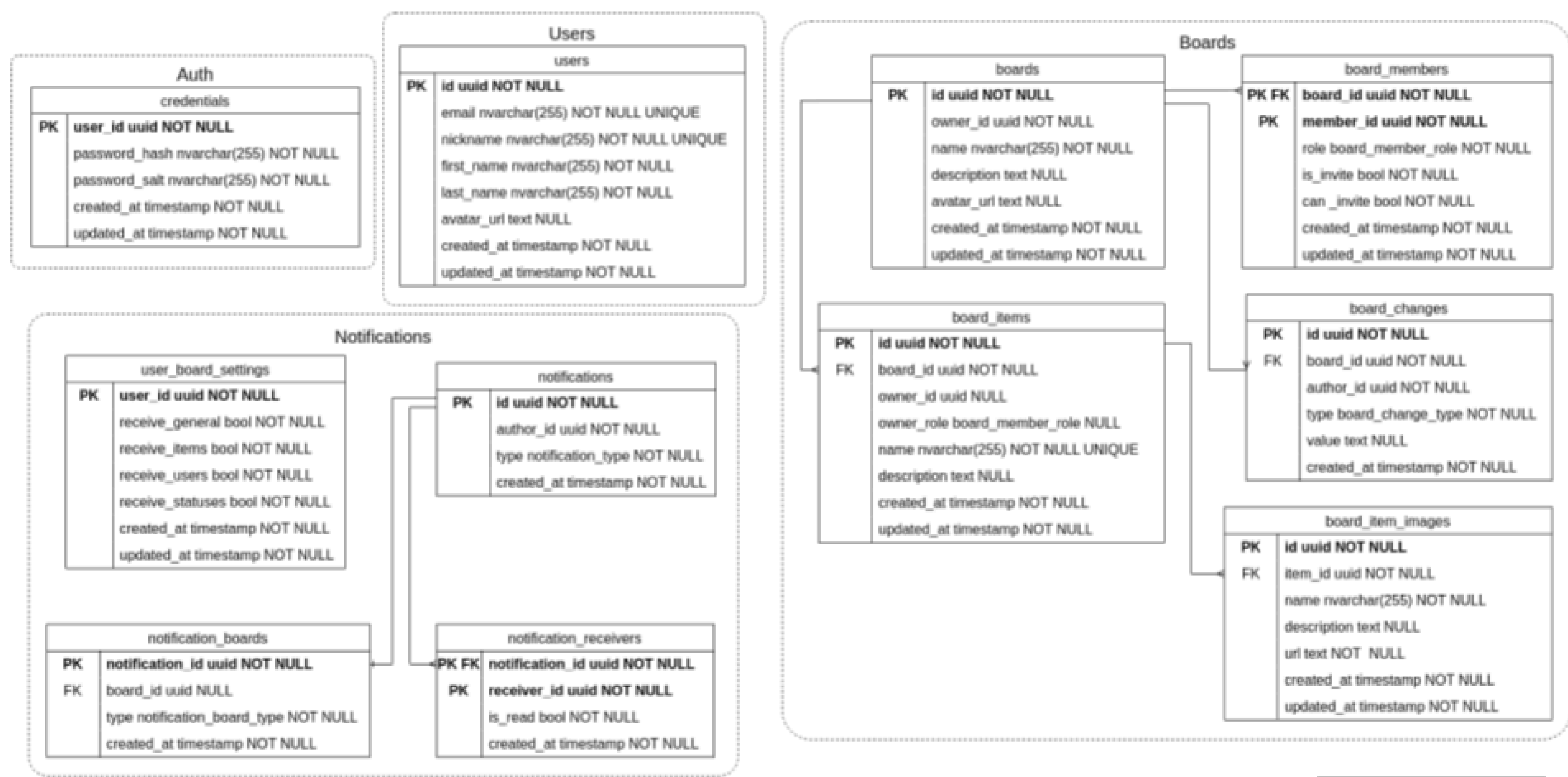


СХЕМА БАЗИ ДАНИХ ЗАСТОСУНКУ



АЛГОРИТМ АДРЕСАЦІЇ ЗОВНІШНІХ ЗАПИТІ

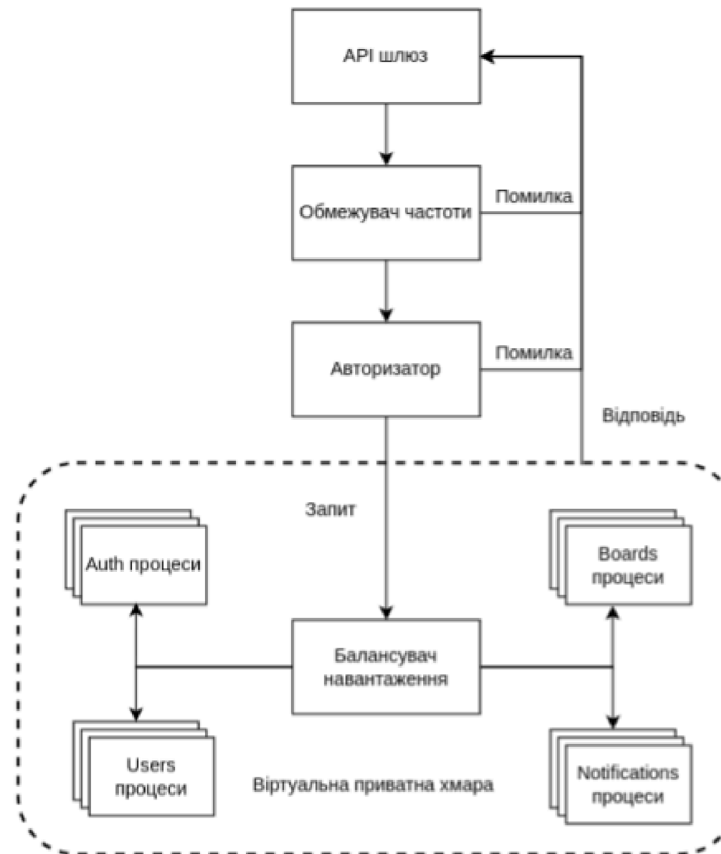
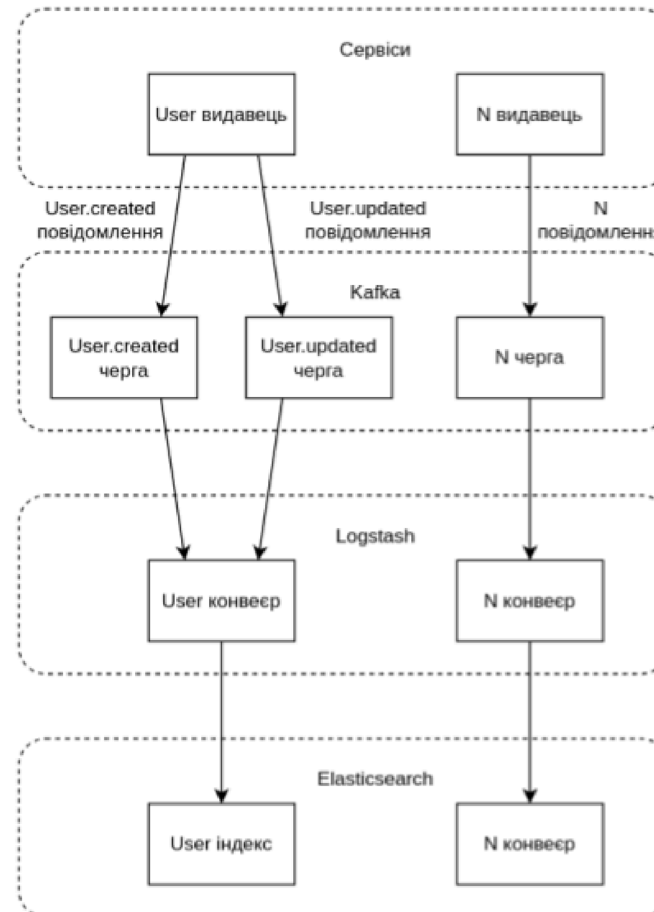


СХЕМА ЕТАПІВ ЛОГУВАННЯ



ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

№	Умови тестового випадку	Очікуваний результат
1	Перебуваючи на сторінці авторизації, користувач вводить пошту та пароль існуючого облікового запису й натискає на кнопку підтвердження.	Застосунок переадресовує користувача на головну сторінка застосунку.
2	Перебуваючи на сторінці авторизації, користувач вводить пошту та пароль, що не збігаються з жодним обліковим записом, і натискає на кнопку підтвердження.	Якщо обліковий запис з введеною поштою не існує – виводиться повідомлення про відсутність знайденого користувача. Якщо обліковий запис з введеною поштою існує, проте не збігається пароль – виводиться повідомлення про неправильний пароль.
3	Перебуваючи на сторінці авторизації, користувач залишає поля пошти та паролю порожніми й натискає на кнопку підтвердження.	Виводиться повідомлення про необхідність заповнення полів.

ПОРІВНЯННЯ З АНАЛОГАМИ

Критерій	Паперові договори	MRI Property Tree Connect	Rental Property Management	Розроблене ПЗ
Зручність користувацького інтерфейсу	–	+	–	+
Безпека та цілісність даних	–	+	+	+
Можливість документування майна	–	–	–	+
Покращення прозорості орендних відносин	+	+	+	+
Можливість співпраці з іншими користувачами	–	+	–	+
Можливість налаштування отримання сповіщень	–	–	+	+

НАПРЯМКИ ПОДАЛЬШОГО РОЗВИТКУ ПРОДУКТУ

- Впровадження оновлення даних у реальному часу.
- Розробка системи монетизації з базовою безкоштовною версією та тарифними планами.
- Інтеграція чатів дошок для обговорень.
- Створення модуля аналітики та візуалізованих звітів про зміни стану майна.
- Додавання багатомовної підтримки та роботи з різними валютами для міжнародних користувачів.
- Реалізація офлайн-режиму з синхронізацією при відновленні інтернет-з'єднання.

ВИСНОВКИ

- Досліджено існуючі рішення у сфері управління майном та виявлено їхні недоліки.
- Сформульовано вимоги програмного забезпечення.
- Обрано технологічний стек та обґрунтовано його доцільність.
- Спроєктовано архітектуру застосунку з гнучким розподілом відповідальності між компонентами.
- Імплементовано заплановані функціональні можливості системи.
- Проведено комплексне тестування з використанням автоматизованих та ручних методів.
- Визначено перспективні напрямки розвитку продукту та можливості масштабування.

ДЯКУЮ ЗА УВАГУ

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
________ Євгенія СУЛЕМА
“___” ____________ 2024 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ФІКСУВАННЯ МАЙНА ОРЕНДАРЯ ТА ОРЕНДОДАВЦЯ ПІД ЧАС НАДАННЯ ПОСЛУГ З ОРЕНДИ

Програма та методика тестування

ДП.045440-04-51

“ПОГОДЖЕНО”

Керівник проєкту:

__________ Яків ЮСИН

Нормоконтроль:

__________ Микола ОНАЙ

Виконавець:

____ Дмитро КУЧЕРЕНКО

2024

ЗМІСТ

1. Об’єкт випробувань..3
2. Мета тестування.. 3
3. Методи тестування.. 3
4. Засоби та порядок тестування.. 4

1. ОБ'ЄКТ ВИПРОБУВАНЬ

Програмне забезпечення для фіксування майна орендаря та орендодавця під час надання послуг з оренди, яке являє собою вебсайт, розроблений з використанням таких технологій, як Go, Gin, Next.js, PostgreSQL, Elasticsearch.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) працездатність компонентів вебсторінок та їхня реактивність;
2) маршрутизація користувацької частини застосунку;
3) зручність та адаптивність користувацького інтерфейсу;
4) приватність доступу до бази даних;
5) захищеність маршрутів серверу;
6) відповідність вимогам технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Протягом усього циклу розробки застосунку було впроваджено систематичний підхід до контролю якості програмного продукту. Це забезпечило своєчасне виявлення та усунення дефектів на початкових етапах, що значно підвищило ефективність розробки та зменшило витрати на подальше обслуговування системи.

Для забезпечення якості програмного забезпечення було реалізовано багаторівневу стратегію тестування, що включала як автоматизовані, так і ручні методи перевірки. Для серверної частини створено перелік автоматизованих тестів доменної логіки, які запускалися автоматично при кожному оновленні коду для стабільності безперервної інтеграції. Це дозволило контролювати коректність роботи бізнес-правил та запобігати

регресіям при внесенні змін.

Паралельно проводилося ручне тестування інтерфейсів за допомогою спеціалізованих інструментів, зокрема створено детальну колекцію тестових запитів. Кожен маршрут перевірявся на відповідність специфікації, коректність обробки різних типів вхідних даних та відповідей сервера.

Особливу увагу приділено тестуванню користувацького інтерфейсу через відтворення реальних сценаріїв взаємодії. Було проаналізовано типові дії користувача та створено відповідні сценарії. При виявленні не очевидної для клієнта поведінки або проблем при використанні проводилося усунення виявлених проблем та повторна перевірка виправленої функціональності.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність програмного забезпечення перевіряється шляхом:

1) статичного тестування коду;
2) автоматизованого тестування реалізації логіки сервісів;
3) автоматизованого тестування серверу при високому навантаженні;
4) ручного тестування користувацьких форм використовуючи граничні та недопустимі значення;
5) ручного тестування на відповідність функціональним вимогам;
6) ручного тестування адаптивності у різних браузерах.

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

__________ Євгенія СУЛЕМА

“___” ____________ 2025 р.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ФІКСУВАННЯ МАЙНА ОРЕНДАРЯ ТА ОРЕНДОДАВЦЯ ПІД ЧАС НАДАННЯ ПОСЛУГ З ОРЕНДИ

Керівництво користувача

ДП.045440-05-34

“ПОГОДЖЕНО”

Керівник проєкту:

__________ Яків ЮСИН

Нормоконтроль:

__________ Микола ОНАЙ

Виконавець:

______ Дмитро КУЧЕРЕНКО

2025

ЗМІСТ

1. Опис структури програмного забезпечення..3
2. Аутентифікація та авторизація...3
3. Сторінка дошок..5
4. Сторінка сповіщень.. 10
5. Сторінка налаштувань...11

1. Опис структури програмного забезпечення

Програмне забезпечення для фіксування майна складається з вебсторінок, котрі відображаються динамічно в результаті взаємодії користувача з вебсайтом.

Вебзастосунок складається із таких сторінок:

1. Сторінка аутентифікації та авторизації облікового запису.
2. Сторінка керування дошками.
3. Сторінка отриманих сповіщень.
4. Сторінка налаштувань профілю та сповіщень.

Авторизувавшись в системі, користувач буде переадресований на список з власними дошками.

2. Аутентифікація та авторизація

Для ідентифікації облікового запису необхідно авторизуватись в системі, інакше користувач буде переадресований на відповідну сторінку. Для цього потрібно зазначити пошту та пароль існуючого запису (рис. 1) та натиснути кнопку «Sign in». При успішній обробці запиту користувач буде переадресований до головної сторінки.

У випадку введення некоректних облікових даних система відображає повідомлення про помилку, яке допомагає користувачу зрозуміти причину невдачі. Система відображення помилки під час помилковій дій реалізована для кожної форми даних застосунку. Для нових користувачів передбачено зручне посилання «Sign up new account», яке переспрямовує на сторінку аутентифікації. Форма входу підтримує автозаповнення полів браузером для швидшого доступу до системи.

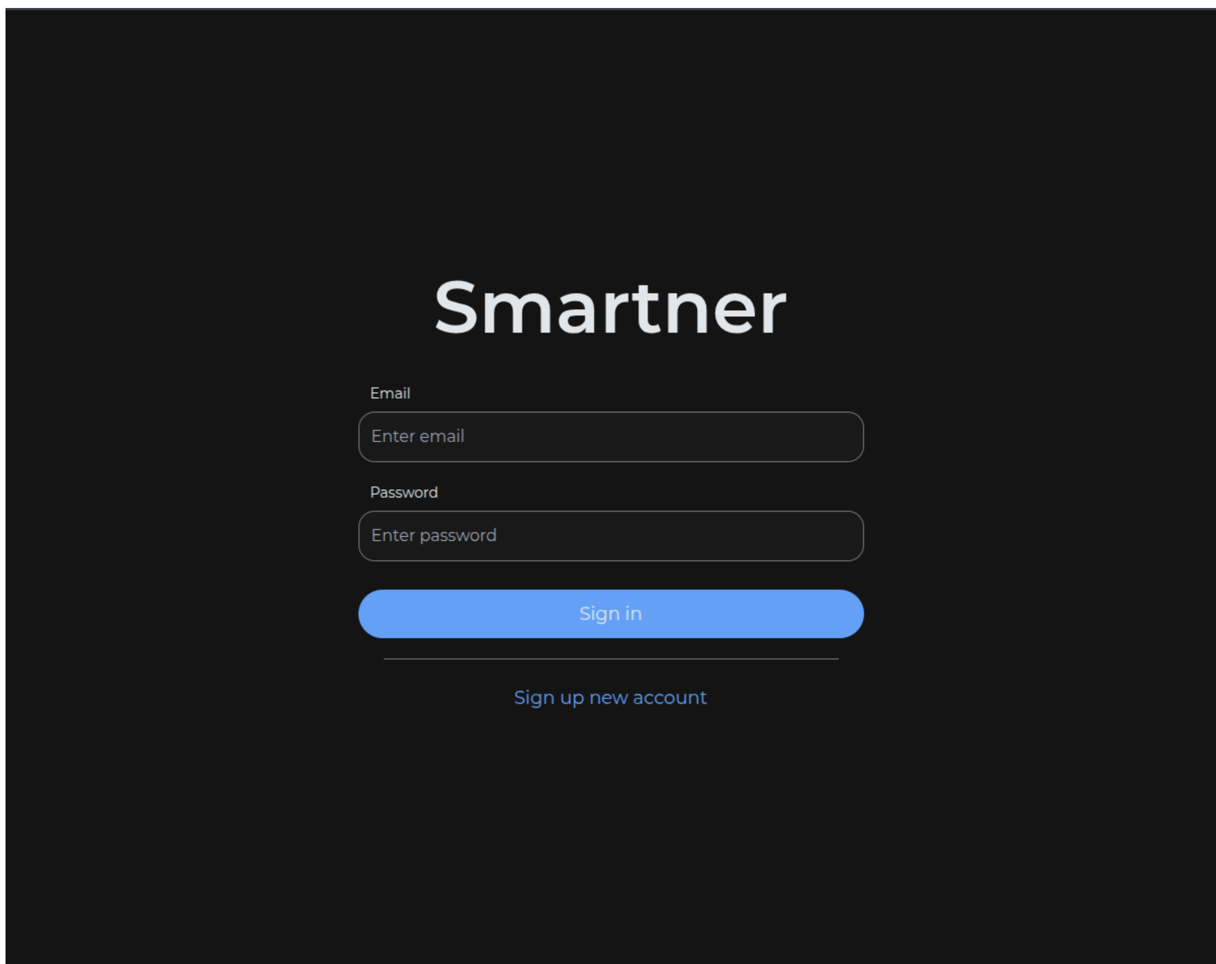


Рис. 1. Сторінка авторизації

Якщо користувач вперше потрапив до вебзастосунку, він може створити новий обліковий запис, натиснувши на кнопку «Sign up new account» та перейшовши до сторінки аутентифікації (рис. 2).

Для створення облікового запису необхідно зазначити ім’я та прізвище, унікальний нік, електронну пошту та пароль і натиснути на кнопку «Sign up». При успішній обробці запиту користувач буде переадресований до головної сторінки.

Для користувачів, які вже мають обліковий запис, передбачено зручне посилання «Sign in to existing account», яке переспрямовує на сторінку авторизації.

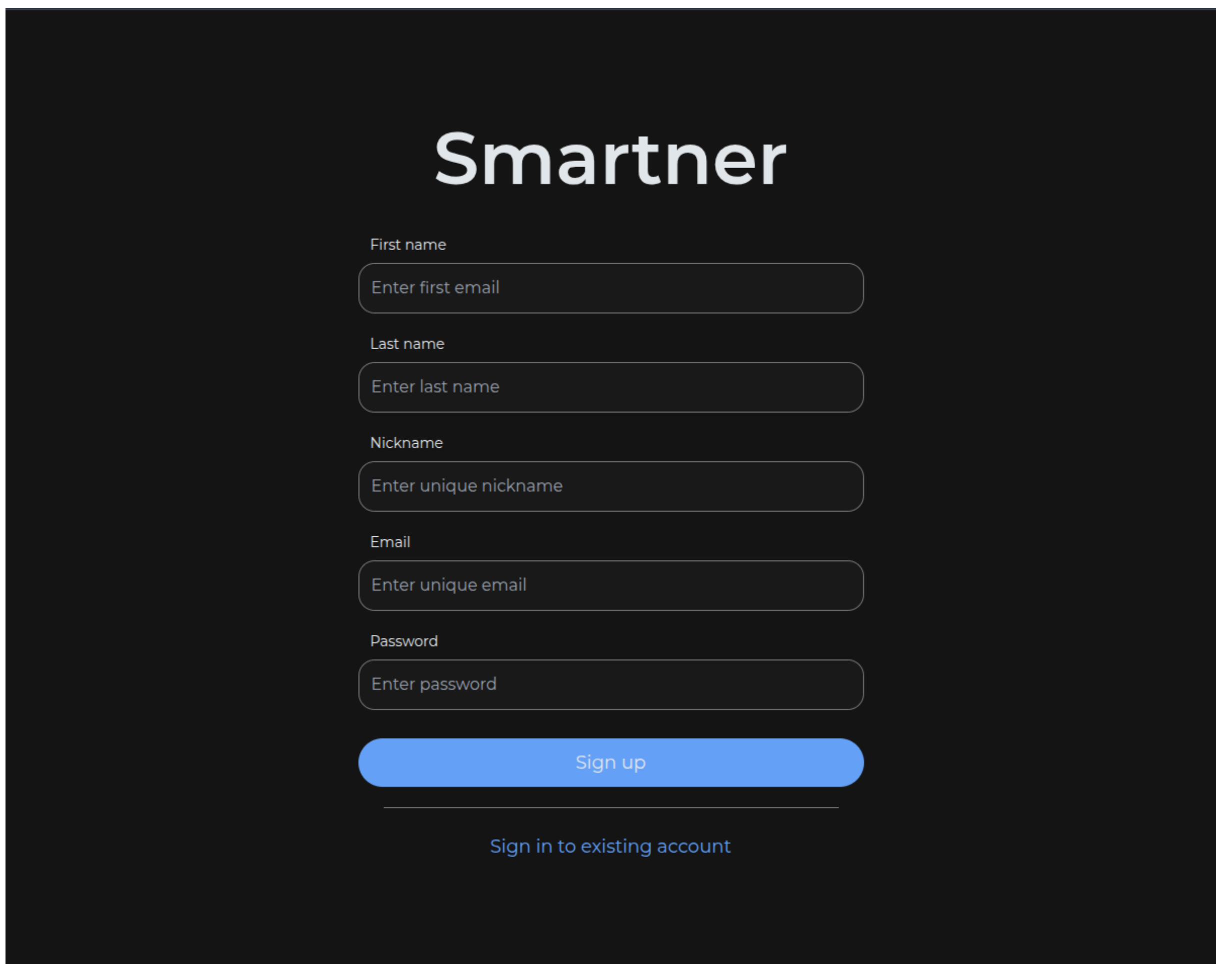


Рис. 2. Сторінка аутентифікації

3. Сторінка дошок

На кожній сторінці для авторизованих користувачів відображається навігаційне меню, яке включає перехід до головної сторінки, сторінки сповіщень та сторінки налаштувань. На головній сторінці відображено список наявних дошок приміщень користувача (рис. 3). Цей список можливо фільтрувати за текстовим рядком, який використовується для пошуку за назвою або описом, сортувати за датою створення та оновлювати список. Опція оновлення даних присутня на кожній сторінці.

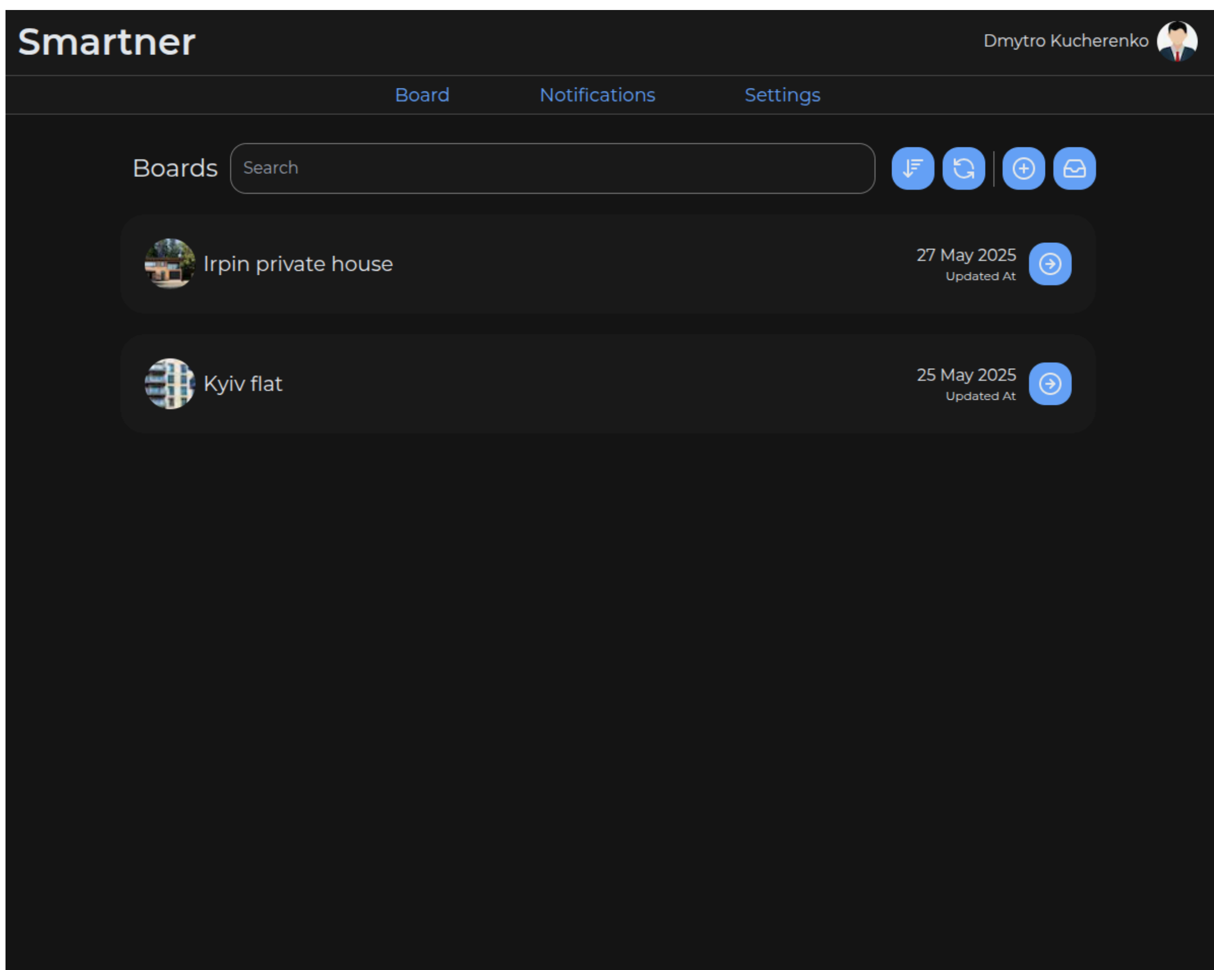


Рис. 3. Сторінка списку дошок

Поруч з полем пошуку можна переглянути можливі дії зі списком. Натиснувши на кнопку з іконкою плюса та перейшовши до створення дошки, відкриється модальне вікно, де необхідно зазначити дані дошки, такі як назва, опис та загальне зображення, та натиснути на кнопку «Submit».

Натиснувши на кнопку з іконкою вхідної скриньки та перейшовши до списку запрошень, користувач буде переадресований на відповідну сторінку (рис. 4), де відображено список запрошень до дошок, які були надіслані іншими користувачами. Кожне запрошення можливо прийняти та відхилити. При прийнятті користувач буде переадресований на сторінку дошки.

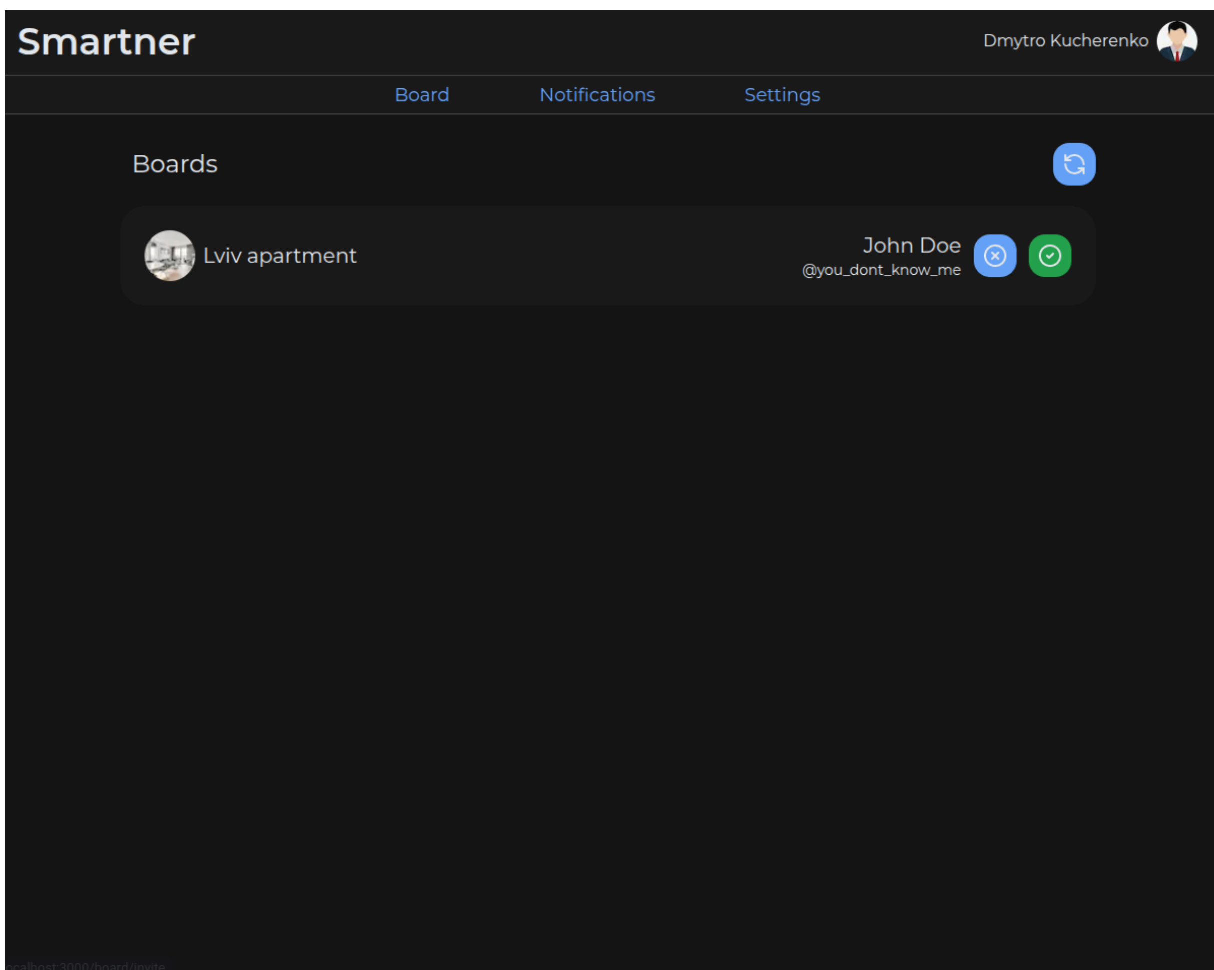


Рис. 4. Сторінка списку запрошень

При натисканні на певну дошку зі списку користувач буде переадресований на її сторінку, де відображено список майна дошки (рис. 5). Його також можна фільтрувати на текстовим рядком та сортувати за датою оновлення.

Параметри для ідентифікації дошки зберігається в URL-адресі сторінки, що дозволяє ділитися посиланнями конкретної дошки з іншими учасниками. Це особливо корисно при обговоренні певної категорії предметів або при необхідності звернути увагу осіб на певний перелік майна. При переході за таким посиланням інші користувачі одразу бачать ту ж саму вибірку предметів.

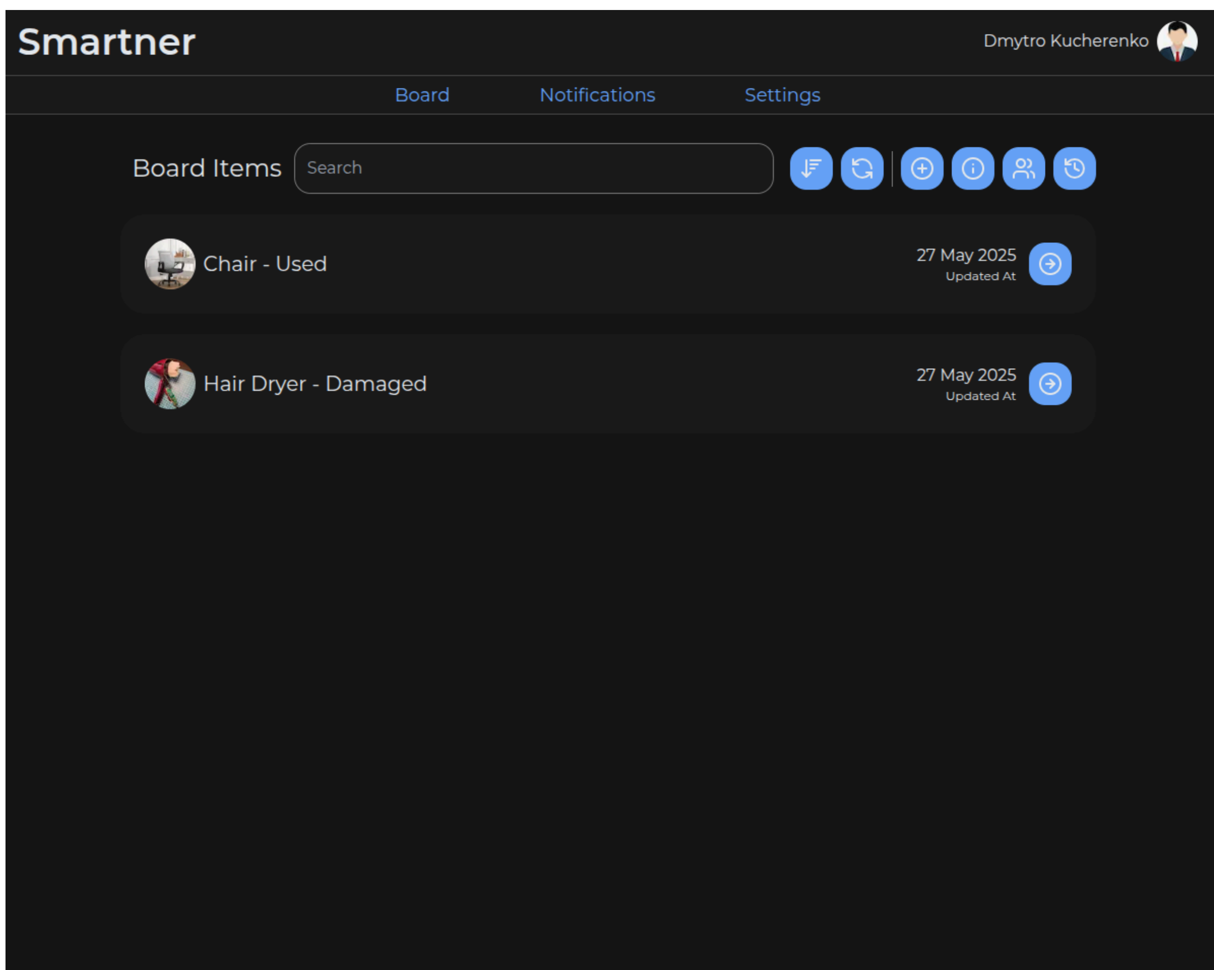


Рис. 5. Сторінка списку майна дошки

Натиснувши на кнопку з з іконкою інформації та перейшовши до перегляду дошки, відкриється подібне до створення дошки модальне вікно, де можна переглянути загальну інформацію та редагувати її, перейшовши за опцією «Edit», змінивши певні поля та натиснувши на кнопку «Submit», або видаляти майно, натиснувши на кнопку «Remove».

При натисканні на кнопку з з іконкою стрілки біля певного майна зі списку відкриється модальне вікно, де можливо переглянути дані про майно, редагувати цю інформацію, перейшовши за опцією «Edit», змінивши певні поля та натиснувши на кнопку «Submit», або видаляти майно, натиснувши на кнопку «Remove».

Натиснувши на кнопку з іконкою плюса та перейшовши до створення майна, відкриється подібне до редагування модальне вікно з ідентичними

полями даних. Натиснувши на кнопку з з іконкою користувача та перейшовши до списку учасників, користувача буде переадресовано на відповідну сторінку (рис. 6) з позначкою, що відображає, чи учасник уже прийняв запрошення.

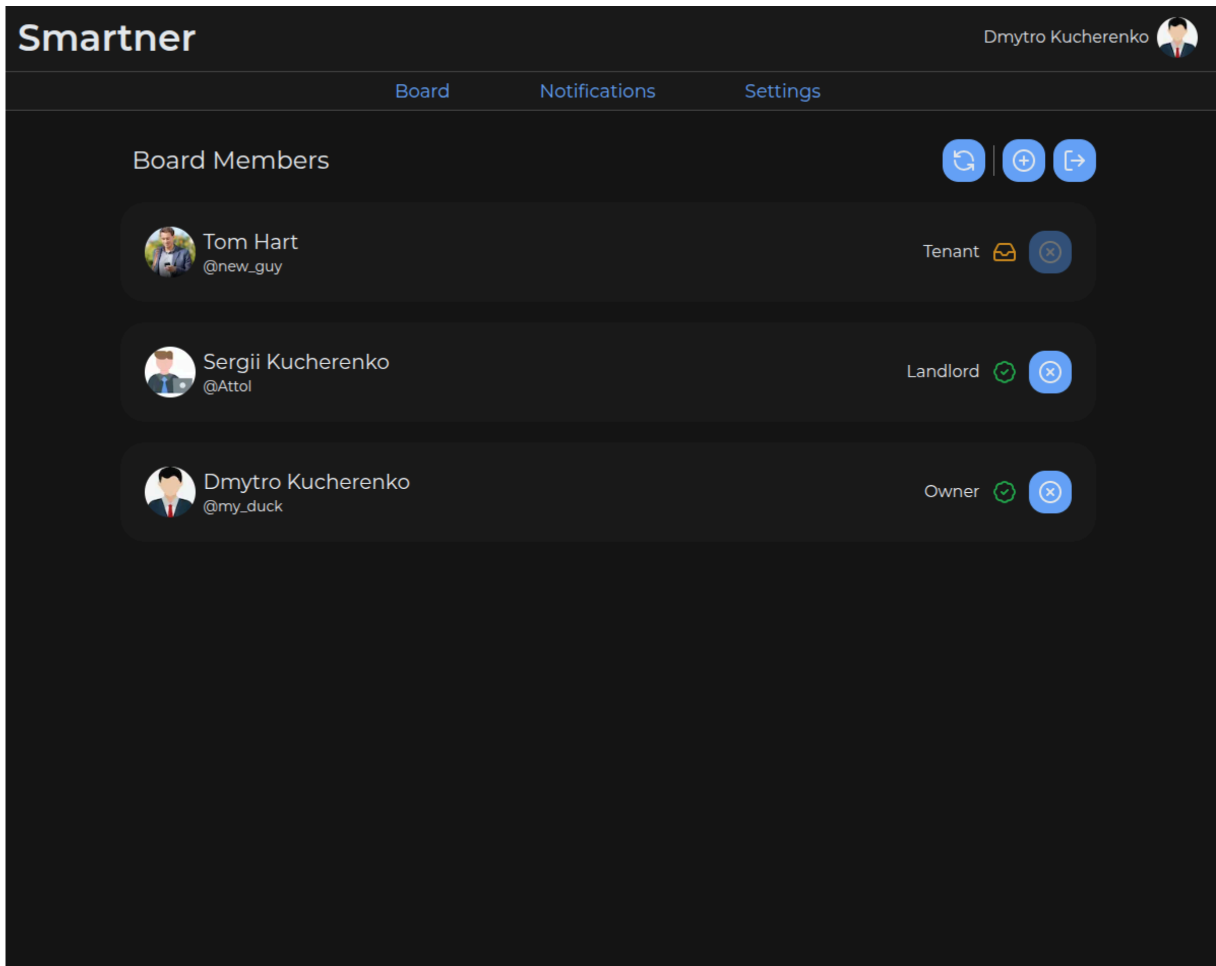


Рис. 6. Сторінка списку учасників дошки

Відповідно до своєї ролі та її прав користувач може натиснути на кнопку з іконкою плюса для відкриття модального вікна для запрошення іншого користувача, редагувати ролі доданих учасників та прибирати користувачів з дошки, використовуючи відповідні кнопки та меню вибору біля кожного користувача.

При поверненні до списку дошок, натиснувши на кнопку з іконкою повернення в часі та перейшовши до історії змін, користувача буде

переадресовано на сторінку зі списком змін дошки та її майна (рис. 7). Це дозволяє переглянути життєвий цикл майна та дізнаватися про зміни, що були внесені іншими учасниками.

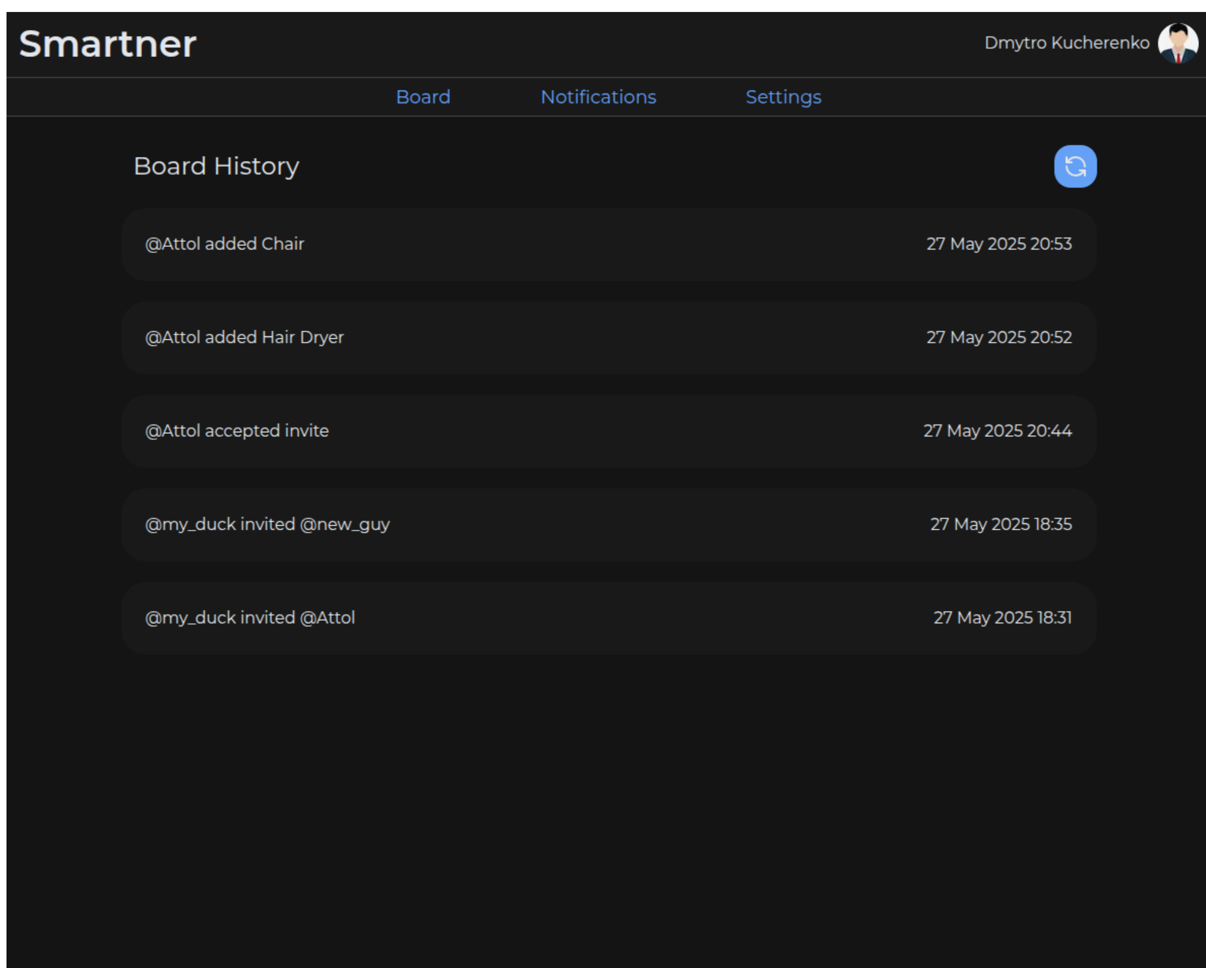


Рис. 8. Сторінка історії змін дошки

4. Сторінка сповіщень

На сторінці сповіщень відображено список персональних сповіщень користувача (рис. 8), які можуть включати сповіщення дошок, запрошення та інші. Нові сповіщення підсвічуються, проте користувач їх завжди може позначати як прочитані за допомогою кнопки з іконкою ока та прибрати за допомогою кнопки з іконкою смітника.

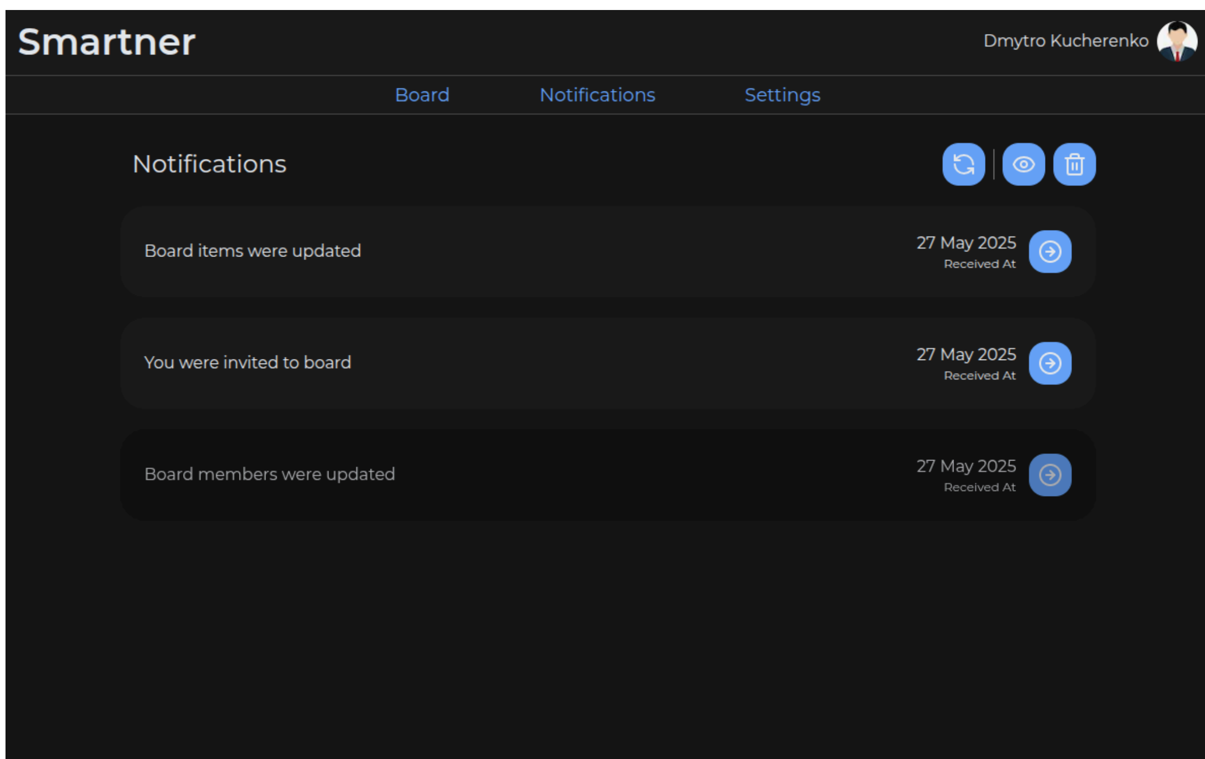


Рис. 8. Сторінка списку сповіщень

5. Сторінка налаштувань

Сторінка налаштувань поділена на розділи, з якими можна взаємодіяти окремо. Перший розділ відповідає за налаштування сповіщень (рис. 9), де можливо увімкнути та вимкнути отримання сповіщень певного типу. Варто зазначити, що деякі сповіщення є обов'язковими для отримання, такі як запрошення.

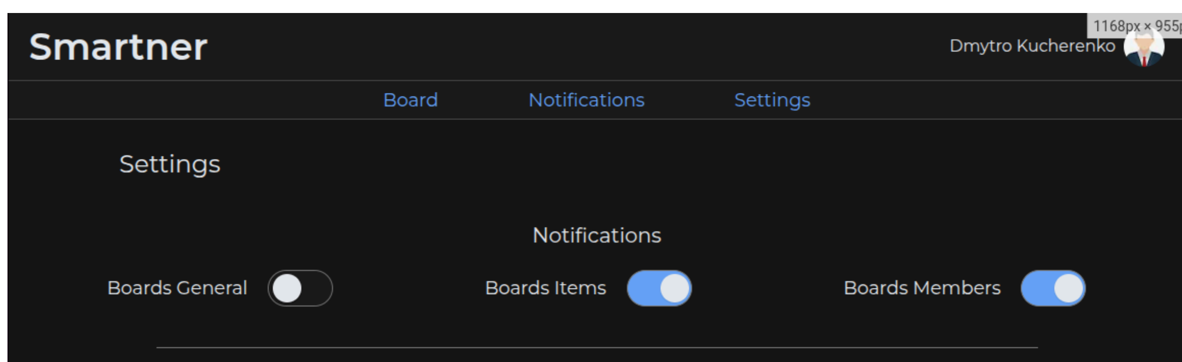


Рис. 9. Розділ сповіщень сторінки налаштувань

Другий розділ надає можливість редагувати ім'я та прізвище облікового запису та змінювати аватар (рис. 10). Після зміни певних даних необхідно натиснути на кнопку «Submit» для підтвердження.

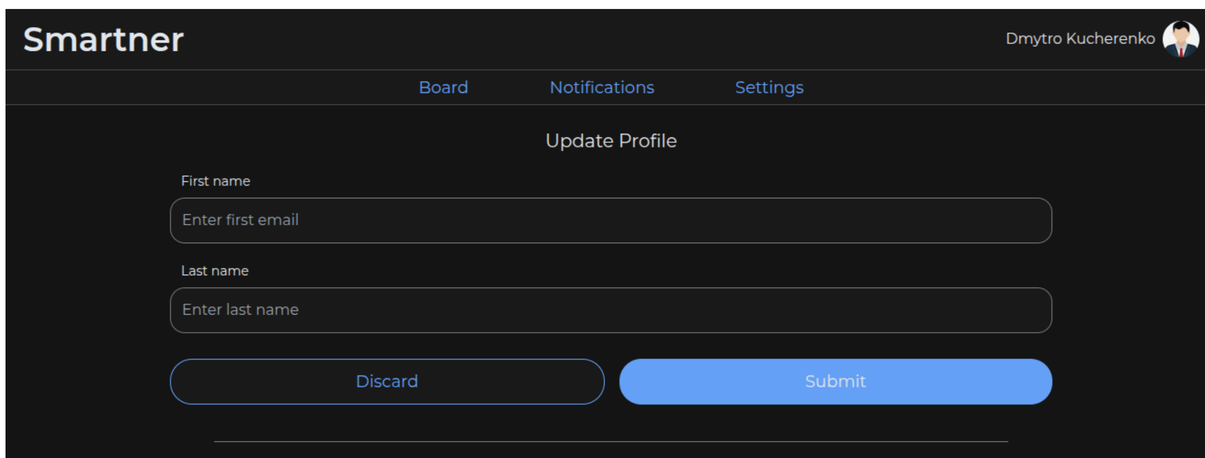


Рис. 10. Розділ оновлення профіль сторінки налаштувань

Останній розділ дозволяє редагувати пароль облікового запису (рис. 11). Для успішної зміни необхідно ввести поточний пароль та новий, після цього натиснути на кнопку «Submit» для підтвердження.

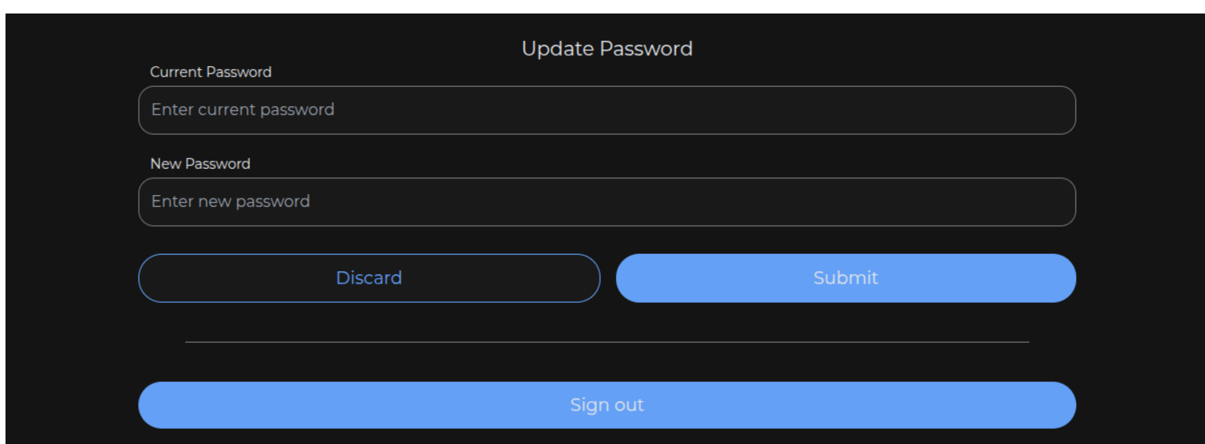


Рис. 11. Розділ оновлення паролю сторінки налаштувань