МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК, ФІЗИКИ ТА МАТЕМАТИКИ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Розроблення інформаційної системи "Smart University"

Кваліфікаційна робота (проєкт)
на здобуття ступеня вищої освіти «магістр»

Виконав: здобувач 2 курсу 12-241М групи
Спеціальності: 121 Інженерія програмного забезпечення
Освітньо-професійної програми: «Інженерія програмного забезпечення» другого (магістерського) рівня вищої освіти
Войченко Владислав Володимирович

Керівники к. пед. н., доцент Вінник М.О.; к. пед. н., доцент Єрмолаєв В.А.
Рецензент Кльонов Д. М.
фріланс, full stack developer

Івано-Франківськ – 2024

ЗМІСТ

ВСТУП

Актуальність роботи

Актуальність роботи полягає в потребі керівництва великих та комплексних будівель, розрахованих на одночасне знаходження в них великої кількості людей та в одночасному виконанні різноманітних функцій, в оперативній ідентифікації та вирішенню проблем, пов’язаних з умовами перебування в будівлі та в безпеці відвідувачів.

В особливості ця потреба стосується навчальних закладів які розраховані на одночасне розміщення декількох тисяч людей різних сфер діяльності, серед яких можуть бути представники сфер, в яких дотримання вимог стану приміщень є критично важливим, наприклад в наукових лабораторіях, в яких від стану приміщення можуть залежати результати досліджень або експериментів.

Також навчальні заклади, містять в собі технічні приміщення, які відвідуються не часто, через що ідентифікація проблеми може зайняти великий проміжок часу, що може призвести до небезпечних для життя аварійних ситуацій – пожеж, переохолоджень, збільшенню рівню стресу та загострення хронічних захворювань від завеликого рівню вуглекислого газу у відвідувачів тощо.

Крім того існує проблема комунікації між відвідувачами та персоналом, яка полягає у відсутності можливості оперативно повідомити про проблему через фізичну віддаленість та відсутність способів комунікації, через що відвідувачі не зможуть вчасно повідомити персоналу про початок пожежі, через що початок евакуації може затриматись.

Також, через унікальність більшості навчальних будівель, ускладняється, якщо не унеможливлюється створення універсальних рішень, які б не використовували датчики, технології дистанційної

передачі інформації та технології обробки та подання інформації в зручному для персоналу виді.

Розробка систем з використанням технології інтернету речей, які спрямовані на вирішення подібних проблем є актуальним напрямком, який потребує подальшого розвитку та досліджень[1].

Мета роботи

Мета кваліфікаційної роботи полягає в розробці нової версії проекту «Smart University»[23], який спрямований но моніторинг стану приміщень будівлі, надання інформації про фізичне розташування і опису будівель та на моніторинг фінансових збитків внаслідок обстрілів та інших пошкоджень будівель навчального закладу.

Для реалізації мети кваліфікаційної роботи потрібно:

- Пошук та аналіз схожих проектів
- Аналіз попередньої версії проекту, пошук багів, недоліків, точок розвитку
- Збір вимог та побажань від керівництва навчального закладу
- Створення вимог для нової версії, на основі проаналізованої інформації та вимог від керівництва
- Вибір технологій на основі вимог та потреб
- Опанування технологій за необхідністю
- Створення плану розробки
- Поетапна розробка за створеним планом
- Тестування системи
- Виправлення помилок за наявності

РОЗДІЛ 1
ОГЛЯД ТА АНАЛІЗ ПОПЕРЕДНЬОГО ТА ПОДІБНИХ ПРОЕКТІВ

1.1 Огляд попереднього проекту

Минулий проект складається з фронтенд та бекенд частин та бази даних.

Фронтенд частина містить наступні сторінки[2]:

- Головна
- Інформація про факультети
- Сторінки будівель
- Сторінки поверхів
- Тестову версію сторінок приміщень
- Сторінку входу для адміністратора
- Сторінку створення нового об’єкту (будівля, поверх, факультет, кафедра, приміщення) в базі даних для адміністратора
- Сторінки зі списком всіх об’єктів для адміністратора
- Сторінку з переглядом об’єкту для адміністратора
- Сторінку для редагування об’єкту для адміністратора
    - Бекенд частина містить:
- Систему авторизації користувачів
- Тестову систему ролей користувачів
- Базу даних з усіма об’єктами
- Базу даних з користувачами
- API для отримання даних за типами та ідентифікаторами об’єктів
- API для створення, видалення та редагування об’єктів
    - Використані технології:
- HTML [35], CSS [36], JavaScript [37] для реалізації фронтенд частини

- Бібліотека VueJs [16] для прискорення розробки фронтенд частини
- CSS бібліотека Bootstrap [17]
- Збирач проектів Gulp [34]
- Платформа NodeJs [9] для розробки серверної частини
- База даних MongoDB [18]
- Бібліотека ExpressJs [10] для генерації сторінок на боці бекенду
- Бібліотека Multer [12] для роботи з файлами
- Бібліотека Mongoose [11] для взаємодії NodeJs та бази даних MongoDB

Вміст сторінок будівель, поверхів та приміщень є динамічним та генерується на бекенд частині, використовуючи інформацію з бази даних. Інші сторінки мають статичну інформацію, яка редагується шляхом зміни змісту ejs файлів.

В системі існує можливість завантажити зображення до опису будівель, а також прикріпити інтерактивний svg об’єкт до сторінки будівлі.

Також система розрахована на можливість імплементації системи користувачів та рівнів доступу, але не має можливості створити нових, та не має функціоналу обмеження доступу за ролями.

Проект має велику кількість «заглушок», які не реалізовані, або реалізовані не повністю.

Сайт має адаптивну верстку для всіх сторінок, що дозволяє користуватись нею на всіх типах пристроїв.

Система дозволяє додавати посилання на датчики до об’єктів приміщень, але не має реалізації збору та відображення інформації з датчиків

1.2 Огляд та аналіз схожих проектів в контексті розвитку проекту

В минулій роботі було розглянуто та проаналізовано перелік рішень розумних університетів, приміщень, будівель, міст в контексті наявного в них функціоналу[2]. На даному етапі розробки варто проаналізувати проекти в контексті UI/UX рішень, та детальної реалізації елементів інтерфейсу, які можуть використовуватись в оновленій версії проекту, для розробки максимально зручних рішень як для звичайних користувачів так і для адміністраторів системи.

З переліку додатків, які будуть аналізуватись, було виключено ті системи, які були реалізовані як прототип, або які не мають дослідницької цінності в контексті розробки нової версії проекту.

Система пошуку шляхів в приміщенні STEERPATH – З попереднього аналізу даний проект зазнав змін, команда розробників переорієнтувала додаток з навчальних закладів на офісні приміщення. В аналізі буде розглянуто як попередню так і нову версії, через відсутність функціоналу для навчальних закладів в новій версії, та через застарілість попередньої версії.

Ця система представляє собою інтерактивну мапу, реалізовану у вигляді вебсайту та додатку для мобільних пристроїв, яка дозволяє персоналу будівель дізнаватись інформацію про приміщення з датчиків, а також бронювати кімнати на певний час, для використання їх в робочих цілях, ця система дозволяє іншим дізнатись, чи зайняте на даний момент приміщення та спланувати своє перебування в будівлі.

Рис. 1.1. Веб-версія оновленого Steerpath

Керівництво та авторизований персонал підприємства має можливість редагувати 3д модель будівлі та додавати та редагувати ній зони та кімнати на спеціальній сторінці на веб-сайті мапи будівлі. Для редагування контенту не потрібно мати спеціалізованої освіти та навичок, через що знаходження спеціалізованого редактора для підтримки сайту є не обов’язковим і дозволяє зекономити на цьому кошти.

Система має заготовлені типи кімнат та об’єктів, які можна налаштувати за власними потребами, встановити їм ім’я, назву, теги, ідентифікатори, іконки.

В минулій версії системи, яка була орієнтована на навчальні заклади були присутня можливість переглянути відвідуваність певних місць, яка збиралась за датчиками, але ця функція була відкинута через дорогу підтримку, необхідність закуповувати та встановлювати датчики, база даних займала місце на жорсткому диску серверів, водночас не користувалась популярністю і не мала великої інформаційної цінності.

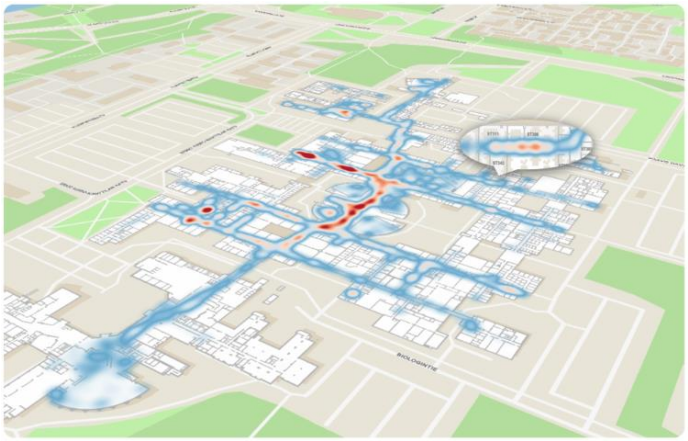

Рис. 1.2. Мапа популярності приміщень в минулій версії Steerpath

Версія для навчальних закладів мала заздалегідь створені типи приміщень, іконку та колір яких змінити було неможливо, водночас існував фільтр, який дозволяв користувачу підсвічувати певні типи приміщень.

Також в минулій версії існувала можливість поділитись координатами приміщення або об’єкту за посиланням, щоб студенти та викладачі могли скоординувати між собою зустрічі. В новій версії цей функціонал відсутній, натомість є список зі співробітниками, їх планами та забронюванні ними приміщеннями.

В новій версії системи фільтрування відбувається за критерієм вільності приміщень, та за наявними зустрічами, які проходять в момент обирання фільтру.

В контексті розробки вимог до нової версії системи розумного університету можна зробити наступні висновки:

- Функціонал відслідковування популярності приміщень має дороге обслуговування та низьку користь.
- Немає необхідності розроблювати систему створення власних типів приміщень, так як університети, зазвичай, мають невелику кількість приміщень з стандартизованим використанням – аудиторії,

лабораторії, кабінети, кафедри, технічні приміщення, коридори, актові зали, столові та вбиральні, тому варто обмежитись стандартизованими об’єктами, що дозволить їх фільтрувати як в панелі адміністратора, так і на користувацькій частині сайту.

- Детальна 3d мапа будівель може черезмірно навантажувати пристрої користувачів, що може унеможливити користування системою певними групами користувачів.
- Орієнтація в просторі за допомогою 3d мапи може бути складною та надлишковою, бо, як правило, певні групи користувачів - представники факультетів, студенти, тощо, зацікавлені в схемі поверхів власного факультету, а не всього університету.
- Розробка можливості легкого редагування схем поверхів персоналом не є необхідною, через не часту зміну структури будівлі, високу ціну розробки редактора, натомість використання svg схем будівель, які можуть бути відредаговані веб-розробниками буде достатньо, для надання необхідної інформації про будівлю, та орієнтації за поверхами.

  В той же час можливість детально змінювати назви, номери, описи приміщень, поверхів та будівель є необхідною, через постійний розвиток навчальних закладів, перепрофілюванням приміщень, зміну факультетів, їх назв, створення нових, зникнення старих. Наявний функціонал вже дозволяє завантажувати svg файли будівлі, але не дозволяє редагувати такі схеми поверхів. В новій версії необхідно доопрацювати можливість додавання та редагування цих схем.

CitiMan[5] – Система розумних міст, націлена на отримання інформації з датчиків, збору інформації, відображення інфографіки, на основі зібраної інформації.

Система дозволяє створювати правила для взаємодії розумних пристроїв та датчиків, наприклад увімкнення освітлення при значеннях освітлення нижче встановлених, або ж увімкнення світла при займанні парковочного місця та недовгий період після.

Інтерфейс реалізований у вигляді веб-сайту з дешбордом, який відображає інформацію у реальному часі.

Інформація з датчиків стану повітря відображається у текстовому форматі, інформація про використання електроенергії подається у вигляді діаграми використання за годинами, розподіл зайнятих, вільних та доступних місць для паркування подається за допомогою кругової діаграми.

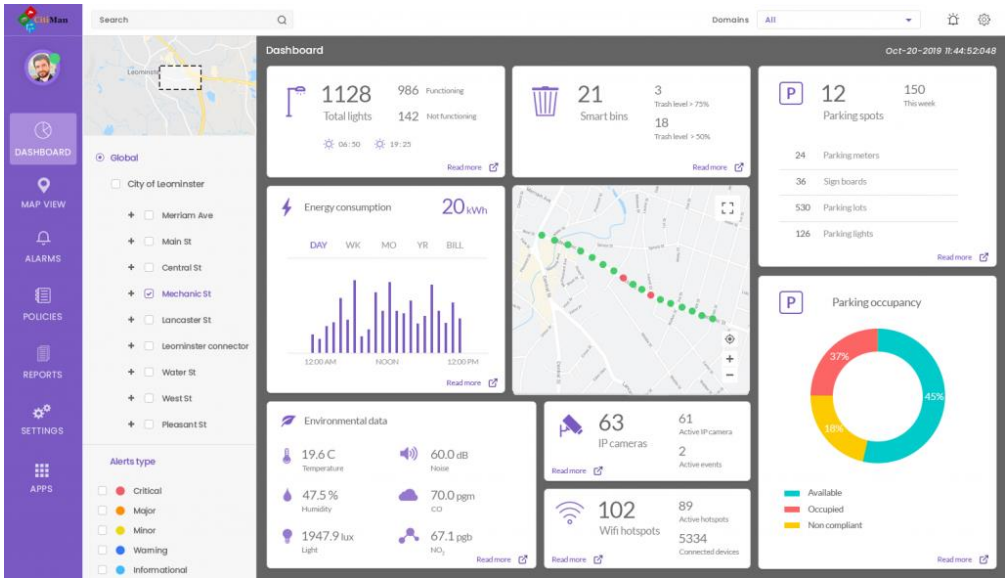

Рис. 1.3. Інтерфейс CitiMan

Система розумного університету вже містить подібну панель адміністратора, реалізовану у вигляді дешборду.

Нова версія проекту розумного університету повинна мати оновлений спосіб подання інформації з датчиків та кількість посадкових місць в приміщеннях.

Реалізувати елементи інтерфейсу, для подання інформації з датчиків можна аналогічним до оглянутого проекту чином – згруповані в один елемент всі дані з датчиків, які складаються з назви величини та її значення.

Реалізувати відображення кількості посадкових місць можна за допомогою діаграми, де на вісі абсцис будуть розміщуватись номери приміщень, а на вісі ординат – кількість посадкових місць. В додаток при наведенні на кожен стовбець – відображати спливаюче вікно з текстовим значенням.

1.3 Пошук точок розвитку проекту, на основі проаналізованих даних

Для подальшого розвитку системи необхідне отримання вимог від керівництва навчального закладу та наукового керівника, обговорення вимог, пошук точок розвитку проекту, пошук багів та проблемних моментів.

На основі обговорень з науковим керівником було визначено, що на даному етапі розробки необхідно сконцентрувати увагу на розробці сторінки зі збитками від обстрілів, виправленню багів та приведенню системи до вимог, сформованих в попередній роботі [3].

При огляді проекту було визначено пріоритетні невідповідності вимогам:

- Відсутність повноцінної системи користувачів з рівнями доступу, персональними акаунтами та можливістю створення ролей
- Відсутність можливості редагувати SVG файл поверху
- Відсутність посилань на залежні кімнати на сторінках поверхів
- Відсутність повноцінної реалізації сторінки кімнати

Додатково було визначено наступні вимоги:

- Статичні сторінки сайту повинні мати змогу редагуватись адміністратором
- Діаграми на сторінках поверхів повинні генеруватись на основі інформації з бази даних
- Сторінка зі збитками повинна мати фільтри за будівлею
- Список на сторінці зі збитками повинен мати функцію сортування за датою та сумою

При плануванні розробки було виявлено, що технології VueJs та ExpressJs не підходять під розмір та складність проекту, та мають структуру, яка ускладнює систематизацію сторінок та об’єктів системи, яка планується розроблятись. Було прийнято рішення про заміну цих технологій на React[24] з використанням TypeScript[38].

При тестуванні проекту було знайдено велику кількість дрібних візуальних багів та невелику кількість функціональних, які необхідно виправити в новій версії проекту.

Також була виявлена несистематичність в файловій структурі – деякі зображення зберігались в папку images, а інші в img.

Було помічено, що минула система не дозволяла створювати поверхи з однаковим номером, але в різних будівлях, цей недолік повинен бути виправлений, для можливості подальшого заповнення бази даних.

Залежні сутності в базі даних не змінювались або не видалялись, при змінах старших об’єктів, що залишало в базі даних велику кількість об’єктів, які не використовуються і які потрібно редагувати або видаляти вручну. В новій версії сервер повинен автоматично редагувати або видаляти такі об’єкти.

В попередній версії деякі сторінки мали незручний або нелогічний інтерфейс та дублювали деякий функціонал, що могло заплутати користувачів.

1.4 Планування розробки нової версії проекту

Перед початком розробки потрібно розробити детальний план, за яким буде вестись розробка, задля систематизації роботи, планування термінів та зменшення витрат часу на визначення цілей під час розробки.

Так як було визначено, що проект повинен мігрувати на React, що потребує зміни структури фронтенд частини та редагуванню певних частин коду, першим ділом потрібно зробити окремий репозиторій на GitHub для нової версії, що дозволить використовувати минулу як архівну і у випадку чого відновити код.

Наступним етапом необхідно створити React проект та поступово перенести фронтенд частину сторінок на нову структуру, виправляючи помічені візуальні баги під час перенесення кожної сторінки.

Далі необхідно переписати бекенд частину під нову бібліотеку – замість генерації сторінок за шаблонами EJS – створити API, яке надсилатиме необхідну інформацію за запитом.

Після цього необхідно переписати старі скрипти на сторінках під новий спосіб взаємодії з бекендом, та написати нові, які робитимуть запити та змінюватимуть дані на сторінках.

Далі необхідно протестувати перенесений сайт та виправити знайдені баги.

Після цього потрібно доробити незавершений функціонал, такий як сторінки кімнат, ролі користувачів, рівні доступу, схеми поверхів, діаграми, тощо. Реалізувавши спочатку бекенд частину, а потім фронтенд.

Аналогічно до попереднього етапу – протестувати та виправити баги.

Далі необхідно реалізувати заплановане розширення функціоналу – сторінка зі збитками, можливість редагувати статичні сторінки, тощо.

Після завершення необхідно протестувати та виправити баги.

РОЗДІЛ 2
ТЕХНІЧНА СПЕЦИФІКАЦІЯ ПРОЕКТУ

2.1 Технології

В оновленій версії проекту будуть використані наступні технології:

Для реалізації фронтенду буде використано популярну бібліотеку React, яка призначена для створення комплексних веб-сторінок з великою модульністю. Також ця система дозволяє перевикористовувати велику частину коду без необхідності переписання компонентів. Мовою програмування було обрано TypeScript, через строгість типізації та самодокументацію коду.

Додатково будуть використані бібліотеки:

- Jest [25] для тестування
- React-router-dom [27] для створення динамічних шляхів до сторінок
- React-cookie [28] для роботи з cookie файлами
- React-hook-form [26] для роботи з формами
- React-slideshow-image [29] для створення каруселей
- Recharts [30] для створення діаграм
- Yup [31] для валідації форм

Для збирання проекту було використано збирач Vite.

Також до перенесення проекту на react буде використано VueJs та ExpressJs для генерації сторінок.

Для серверної частини буде використана платформа NodeJs разом з бібліотекою Mongoose для роботи з базою даних та Multer, для прийому та збереження файлів від користувачів. Для створення API буде використано бібліотеку ExpressJs.

Додатково будуть використані бібліотеки:

- Express-session [33], для реалізації сесій

- Cookie-parser [13], для роботи з cookie файлами
- Cors для роботи з cors
- Jsonwebtoken [15] для реалізації сесій

Для реалізації бази даних буде використано платформу MongoDB. Для тестування, дебагу та редагування бази даних буде використано MongoDBCompass.

Для розробки та дебагу API буде використано програму для ручного створення запитів на сервер – Postman[19].

Для управління версіями та публікації коду буде використано GitHub та GitHub Desktop.

Для демонстрації проекту в умовах відсутності серверу буде використано сервіс ngrok[20], який дозволяє захостити сайт на власному пристрої, та використати тимчасову веб-адресу.

2.2 Реєстр вимог

Реєстр вимог було розроблено в попередніх роботах [2][3]. Його було доповнено наступними вимогами:

- В базі даних повинна бути таблиця зі збитками, кожен запис повинен містити в собі наступну інформацію:
  - Назва збитку
  - Опис збитку
  - Приблизна сума збитку
  - Ідентифікатор будівлі, яка зазнала шкоди
  - Фотографія з фіксацією збитку
  - Стан (очікує, в черзі, в процесі, виконано)
- В дешборді адміністратора повинна бути сторінка зі збитками, яка повинна містити

  - Список збитків, де кожен елемент відображатиме ім’я, корпус та суму
  - Сортування збитків за – датою додавання, сумою, назвою
  - Фільтрування збитків за будівлею та станом
  - Кнопку створення запису про збиток
- Сторінка запису повинна відображати всю інформацію, яка міститься в записі в базі даних
- Сторінка створення запису про збитки повинна мати можливість заповнити кожне поле, яке може існувати до цього типу записів в базі даних
- Інфографіка та діаграми на сторінках поверхів повинні брати інформацію з бази даних
- Головний адміністратор повинен мати змогу створювати нові роли та видавати їх користувачам.
- Головний адміністратор повинен мати змогу надавати наступні можливості ролям
  - Редагування статичних сторінок сайтів
  - Створення записів в базі даних
  - Редагування всіх записів в базі даних
  - Редагування записів в базі даних, пов’язаних лише з певною будівлею
  - Редагування записів в базі даних, пов’язаних лише з певним факультетом
  - Редагування записів в базі даних, пов’язаних лише з певним приміщенням

РОЗДІЛ 3
ТЕХНІЧНА РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Початок міграції проекту на React

В першу чергу було створено новий репозиторій GitHub з дублікатом файлів попередньої версії. Проект було розділено на дві директорії – frontend та backend, де файли в другій були незмінною копією бекенду попередньої версії, а frontend мала стартовий проект React, створений збирачем Vite.

Далі було встановлено та імпортовано необхідні для розробки фронтенду бібліотеки.

Після цього було створено стартову структуру проекту – директорії для компонентів, сторінок, хуків, контекстів, функцій, також було створені файли для типів та стилів.

3.2 Перенесення статичних сторінок та компонентів

Перенесення сайту на нову структуру було почато зі статичних сторінок, які не мають складних скриптів, та слугуватимуть базою для тестування стилів, файлів, скриптів та перших ендпоїнтів API.

Спершу було перенесено головну сторінку – було створено файл Page.tsx зі стандартною структурою react компоненту, в який був доданий html код головної сторінки з попередньої версії.

Наступним кроком було послідовне розрізання файлу на компоненти:

Footer, який містив в собі статичний зміст, який має ідентичній зміст на кожній сторінці сайту, для подальшого використання компоненту для кожної сторінки

Header, який містив статичний контент та навігацію по сайту з підсвічуванням сторінки, на якій знаходиться користувач. Даний елемент

використовуватиметься на всіх сторінках сайту, окрім сторінки авторизації.

Hero – перший екран головної сторінки, який містить заголовок та слайдер з фотографіями університету, та кнопку, яка пропонує перейти до перегляду будівель.

Corpus – елемент посилання на будівлю, який повторюватиметься на сайті, він має іконку будівлі, її назву та посилання на неї.

Також було створено компонент PageScripts, який містив спільні для всіх сторінок скрипти.

Після розбиття сторінки на компоненти було створено Layout для всього сайту, який дозволяє скоротити кількість коду та уніфікувати деякий контент для всього сайту.

До цього компоненту було додано Header, Footer та PageScripts, між першими двома буде вставлятись контент сторінки.

Після цього було написано нову структуру головної сторінки, яка містила в собі лише Layout, компоненти слайдеру, Hero та цикл, який генерував список з компонентів Corpus, який генерувався з тестового масиву, який при подальшій розробці API було замінено на отримання та обробки списку будівель з сервера, за допомогою useState, та useEffect:

```
const [buildings, setBuildings]: any[] = useState([])
…
useEffect(() => {
   const fetchBuildings = async () => {
     const response = await fetch(buildingsAPI)
     const json = await response.json()

     if (response.ok) {
```

```
      console.log(json)

      setBuildings(json)
    } else {
      //TODO: toast error?
      console.log(response.status, response.text)
    }
  }

  fetchBuildings()
}, [])
```

Заголовок в Hero та зображення в слайдері також отримуються з API:

```
const [home, setHome] = useState({
  heading: `Аналітична система матеріально-технічної бази Херсонського Державного Університету`,
  buttonLink: "/buildings",
  images: ["/img/1.jpg", "/img/2.jpg", "/img/3.jpg", "/img/4.jpg"],
})
…
useEffect(() => {
  const fetchHomePage = async () => {
    const response = await fetch(homePageAPI)
    const json = await response.json()

    if (response.ok) {
      console.log("home: ", json)

      setHome(json)
```

```
      } else {
        console.error(response.status, response.text)
      }
    }

    fetchHomePage()
  }, [])
const slides =
    home.images.length > 0
      ? home.images.map((image) => {
          return { url: process.env.PUBLIC_URL + image, caption: "" }
        })
      : [{ url: process.env.PUBLIC_URL + "/img/1.jpg", caption: "" }]
```

Далі було створено статичну сторінку 404, яка відображалась в тому випадку, коли шлях до сторінки не існує в React. Ця сторінка містить в собі код помилки та пропозицію повернутись на головну, оформлену у вигляді посилання.

Наступним кроком було перенесення сторінки з інформацією про факультети. Було розроблено компоненти сітки та інформаційних карток факультетів, які дозволять швидко створювати інформаційні картки про факультети як на сторінці з інформацією про факультети, так в майбутньому і на сторінках поверхів.

3.3 Перенесення сторінки будівель

Для перенесення динамічно згенерованих сторінок на React потрібно, окрім редагування фронтенду, ще змінити API, який би видавав не готову сторінку, а лише JSON файл з інформацією.

Першою було перенесено сторінку зі списком будівель, через свою першочерговість в навігації по будівлі та простій структурі даних.

На першому екрані сторінки було використано карусель, яку було створено на етапі перенесення головної сторінки. Список з будівлями та елементи цього списку було розбито на компоненти, для зручного перевикористання коду.

Зі сторони бекенду було створено файл buildings.js, який містив ендпоїнти для API будівлі – отримання всіх будівель з бази даних у форматі JSON, отримання одної конкретної будівлі за її унікальною назвою. В подальшому подібний код отримання всіх та одного об’єкту буде використовуватись для кожного типу об’єктів:

```
router.get("/", async (req, res) => {
  try {
    const buildings = await Building.find()
    res.json(buildings)
  } catch (err) {
    res.status().json({ message: err.message })
  }
})
//get one
router.get("/:name", getBuilding, (req, res) => {
  res.json(res.building)
})
async function getBuilding(req, res, next) {
  try {
    building = await Building.findOne({
      name: req.params.name,
    })
```

```
    if (building == null) {
      return res.status(404).json({ message: "Can't find building" })
    }
  } catch (err) {
    return res.status(500).json({ message: err.message })
  }
  res.building = building
  next()
}
```

3.4 Перенесення сторінки будівлі

При перенесенні сторінки будівлі її було розділено на декілька компонентів: поверх, паралакс-шапка та компонент, який відображає SVG будівлі.

На боці серверу було створено API поверхів, який дозволяє за ідентифікатором поверху отримувати повну інформацію про нього. Це потрібно для генерації списку поверхів на сторінці будівлі.

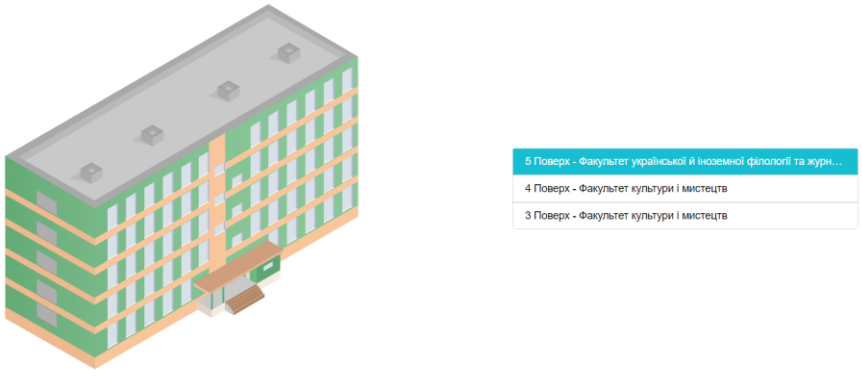

Рис. 3.1. Сторінка будівлі зі згенерованим за інформацією з бази даних списком.

3.5 Перенесення сторінки поверху

Сторінка поверху має одну з найкомплексніших структур в проекті. Її було розбито на наступні компоненти:

- Шапка поверху, яка містить логотип, колір, назву та факультет поверху.
- Мапа поверху, яка містить посилання на svg файл та приймає кольори поверху, для перефарбування svg схеми.
- Дані з датчиків про якість повітря (вуглекислий газ, AQI)
- Дані з датчиків, про стан повітря (температура, вологість)

Інформація про поверх завантажується API поверхів.

Додатково було розроблено API для факультетів, який надає детальну інформацію про факультет, який знаходиться на поверсі.

Елемент з інформацією про факультет було замінено компонентом інформаційної картки про факультет, а поля картки заповнюються отриманою з серверу інформацією.

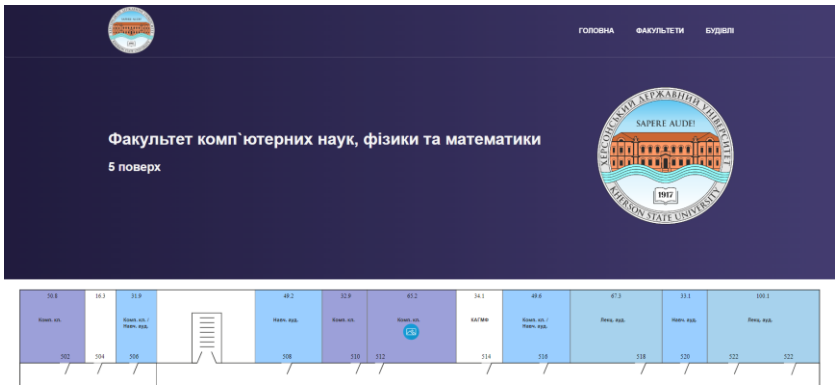

Рис. 3.2. Верхня частина автоматично згенерованої сторінки поверху.

3.6 Перенесення сторінки входу та функціоналу авторизації

Спершу було перенесено сторінку входу в профіль адміністратора. Її фронтенд частина майже не зазнала змін, окрім перероблення скриптів.

Авторизації користувачів на ExpressJs та React значно відрізняються через наявність в React відразу двох різних серверів для бекенду та фронтенду, на відміну від одного в ExpressJs.

В бекенд частині було створено API для системи авторизації:

- Для реєстрації було створено шлях register, який отримує логін та зашифрований пароль користувача. Після цього в коді перевіряється наявність користувача за логіном в базі даних. Якщо користувача не існує, то створюється запис про нового користувача з хешованим паролем, в іншому випадку користувачу надсилається відповідь з помилкою.
- Для входу було створено шлях login, який отримує логін та хешований пароль, якщо в базі даних існує запис з логіном користувача, то за допомогою бібліотеки bcrypt[14] хешований пароль порівнюється з хешованим паролем в базі даних. У випадку, якщо вони співпадають – генерується та надсилається токен авторизації, який діє 24 години.

В фронтенд частині було створено контекст авторизації, який зберігає та керує інформацією про поточного користувача.

```
import { useReducer, createContext } from "react"

export const AuthContext = createContext(null)

export const authReducer = (state: any, action: any) => {
  switch (action.type) {
    case "LOGIN":
      return { user: action.payload }
    case "LOGOUT":
      return { user: null }
    default:
      return state
  }
```

```
}

export default function AuthContextProvider({ children }: { children: any
}) {
  const [state, dispatch] = useReducer(authReducer, {
   user: null,
  })
  console.log("AuthContext state: ", state)
  return (
   <AuthContext.Provider value={{ ...state, dispatch }}>
    {children}
   </AuthContext.Provider>
  )
}
```

Весь контент додатку було загорнуто в контекст авторизації, для можливості отримувати всі дані, та викликати запити на вхід та вихід з будь якої точки сайту.

Також було створено хуки useAuthContext, useLogin, useLogout для зручної роботи с системою авторизації.

Після цього було змінено сторінку входу, та було додано кнопку виходу, які використовували хуки авторизації та виходу з профілю.

Так як дана система не зберігає дані після закриття сторінки, її було модифіковано можливістю завантажувати сесію користувача з кукі файлів. До контексту авторизації було додане записування даних в кукі при отриманні токену з сервера. Також при завантаженні контексту авторизації було додано спробу читання поля токену в файлах кукі, при

існуванні такого поля цей токен привласнювався як значення стану поточного користувача.

В бекенд частині було створено middleware для захищених шляхів API. Він перевіряв кукі файли, надіслані разом з запитом, на наявність дійсного токену користувача. При відсутності такого він повертатиме помилку доступу.

Також для зручності користувача було додано функціонал збереження адреси сторінки, з якої було перенаправлено на сторінку авторизації, та повернення користувача на цю сторінку при успішній авторизації.

3.7 Перенесення сторінки «створення об’єкту» в панелі адміністратора

Панель адміністратора має комплексну структуру, та велику кількість повторюваних елементів на кожній сторінці.

В першу чергу було створено Layout панелі адміністратора, який буде містити в собі наступні компоненти:

- Хедер, який має завжди однаковий стан «з білим фоном та темним логотипом»
- Сайдбар зі сторінками дешборду
- Контейнер для контенту
- Футер

Далі було перенесено зміст сторінки «створити об’єкт», налаштовано форми та створено прототип скрипта відправки запиту на створення об’єкту, який при створенні адміністраторських API для кожного типу об’єкту буде доповнено можливістю відправляти запити на створення цих об’єктів.

3.8 Створення універсальної сторінки для списків

При аналізі існуючого коду та пошуку оптимальної структури, було прийнято рішення створити одну універсальну сторінку для списків, яка б змінювала свій контент в залежності від адреси, за якою до неї звертаються.

Так, при адресі /rooms/ вона робитиме запит на адміністраторський API кімнат.

Було створено прототип скрипта для створення запитів на API, в якому в подальшому будуть реалізовуватись функції запитів, по мірі створення API об’єктів.

Сторінку було розбито на такі компоненти, як:

- Шапка списку, яка міститиме прототипи налаштування сортування та фільтрації інформації
- Тіло списку, яке міститиме в собі контент
- Елементи списку, які відображають дані з об’єкту, який в них передається, та при натисканні перенаправляють користувача за посиланням, яке надається в об’єкті

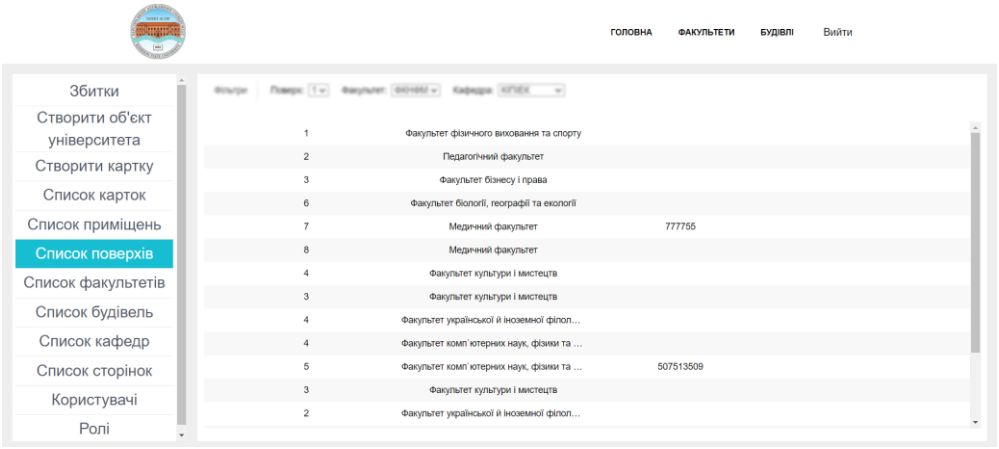

Рис. 3.3. Сторінка зі списком

3.9 Створення API з CRUD функціоналом для кожного об’єкту

Для подальшого перенесення панелі адміністратора потрібно створити повноцінне API для кожного типу об’єктів, яке надаватиме можливість створювати, читати, редагувати та видаляти об’єкти. Також ці API повинні надавати доступ лише авторизованим користувачам.

Отримання об’єкту з бази даних має аналогічний користувацькому ендпоїнту запиту.

Для кожного типу об’єктів було створено окремий ендпоїнт створення з валідацією структури та вмісту, відповідно до типу об’єкту.

Кожен ендпоїнт перевіряє існування копії об’єкту в базі даних за ключовим полем або за декількома полями, в залежності від типу об’єкту.

Далі система робить спробу створити об’єкт в базі даних, у випадку успішного створення вона повертає відповідь про успішне створення, в іншому випадку – повертає помилку.

Ендпоїнти редагування шукають відповідний об’єкт за ключовим рядком або ідентифікатором.

Потім відбувається послідовна перевірка полів на існування, та формується новий об’єкт на основі значень з полів. Якщо поля в запиті не існує, то залишається попереднє. Після цього відбувається запит на редагування об’єкту в базі даних та повертається повідомлення про успішність або помилка, в залежності від результату.

Ендпоїнти видалення шукають об’єкти за ключем або ідентифікатором, при наявності об’єкту – робиться запит на видалення та повертається повідомлення про успішне виконання або помилка, в залежності від результату. При не існуванні об’єкту користувачу повертається відповідна помилка.

Приклад захищеного ендпоїнту створення поверху, який валідує дані та надсилає відповідь про успішність виконання операції:

```
router.post("/", requireAuth, upload.single("svg"), async (req, res) => {
  if (isEditor(req.role)) {
    let isFacultyExists = await Faculty.exists({ name: req.body.faculty })
    let isBuildingExists = await Building.exists({ name: req.body.building
})
    let roomsArray = []

    if (req.body.rooms == null || req.body.rooms == undefined) {
      req.body.rooms = []
    }

    req.body.rooms.forEach((room) => {
      roomsArray.push(Room.exists({ number: room }))
    })

    let isRoomsExists = roomsArray.every((i) => i === true)

    let svg

    if (req.file) {
      svg = req.file.filename
```

```
        }

        if (isFacultyExists && isRoomsExists && isBuildingExists) {
          const floor = new Floor({
            number: req.body.number,
            faculty: req.body.faculty,
            rooms: req.body.rooms,
            building: req.body.building,
            temperatureSensorURL: req.body.temperatureSensorURL,
            co2SensorURL: req.body.co2SensorURL,
            floorColor: req.body.floorColor ?? "#ffffff",
            svg: svg,
          })

          try {
            const newFloor = await floor.save()

            const faculty = await Faculty.findOne({ name: req.body.faculty })
            faculty.floors.push(req.body.number)
            const updatedFaculty = await faculty.save()

            const building = await Building.findOne({ name: req.body.building
})
```

```
            building.floors.push(req.body.number)
            const updatedBuilding = await building.save()

            res.status(201).json({
              floor: newFloor,
              faculty: updatedFaculty,
              building: updatedBuilding,
            })
          } catch (err) {
            res.status(400).json({ message: err.message })
          }
        } else {
          res.status(400).json({ message: "Invalid building or faculty" })
        }
      } else {
        res.status(400).json({ message: "Not enough rights" })
      }
    })
```

API було протестовано за допомогою утиліти Postman, яка дозволяє створювати запити вручну, та MongoDBCompass, для перегляду записів в базі даних.

3.10 Перенесення сторінки створення об’єктів

Наступним кроком було перенесено сторінку створення об’єктів, її функціонал не зазнав змін з попередньої версії.

Суцільний файл з кодом було розбито на компоненти, які представляли з себе форми до відповідних типів об’єктів, при цьому перемикач типів було винесено в окремий компонент, так як він повинен бути незмінним, відносно обраного типу об’єктів.

В залежності від обраного типу користувачу відображаються відповідні форми, а при натисканні на кнопку «створити» - відбувається попередня валідація даних та перевірка на заповненість обов’язкових полів. У випадку помилки – користувачу виводиться відповідне повідомлення, або підсвічується те поле, яке обов’язкове до заповнення.

Далі валідація відбувається вже на боці API, на випадок, якщо вона не спрацює на фронтенд частині, або зловмисник спробує надіслати викривлені дані створивши власний запит в обхід веб-форми.

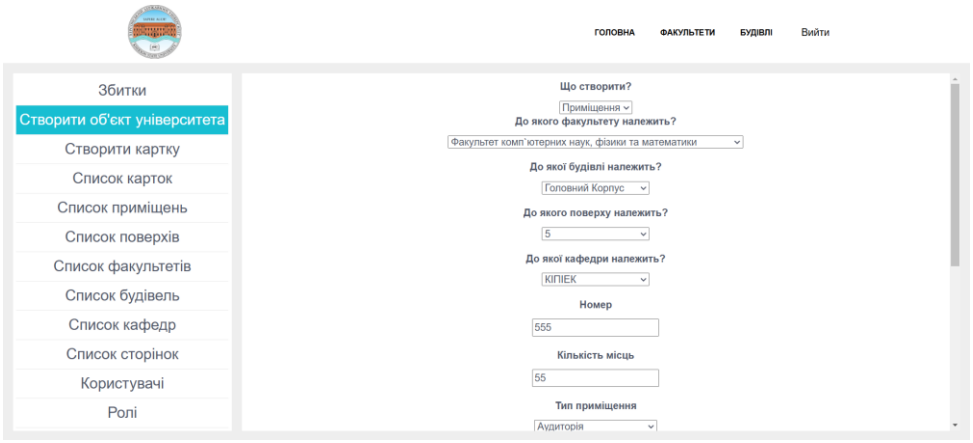

Рис. 3.4. Сторінка створення об’єктів

3.11 Перенесення сторінок перегляду об’єктів

Сторінки перегляду об’єктів мають примітивну структуру – посилання на редагування сторінки, кнопка, яка надсилає запит на видалення об’єкту з бази даних та всі поля.

Для елементів, які повторюються можна створити окремий компонент, в який буде передаватись ідентифікатор та тип об’єкту, на основі яких відбудеться генерація посилання на сторінку редагування та буде згенеровано запит на редагування об’єкту.

Інші поля можуть мати унікальну структуру, тому для кожного типу об’єктів їх буде створено вручну.

При переході за посиланням, яке має структуру «/назва-списку/ідентифікатор» буде відправлятись запит на сервер з відповідним типом та ідентифікатором. У випадку наявності запису – генерується та відображається сторінка, у випадку неіснування – відбудеться переадресація на сторінку 404.

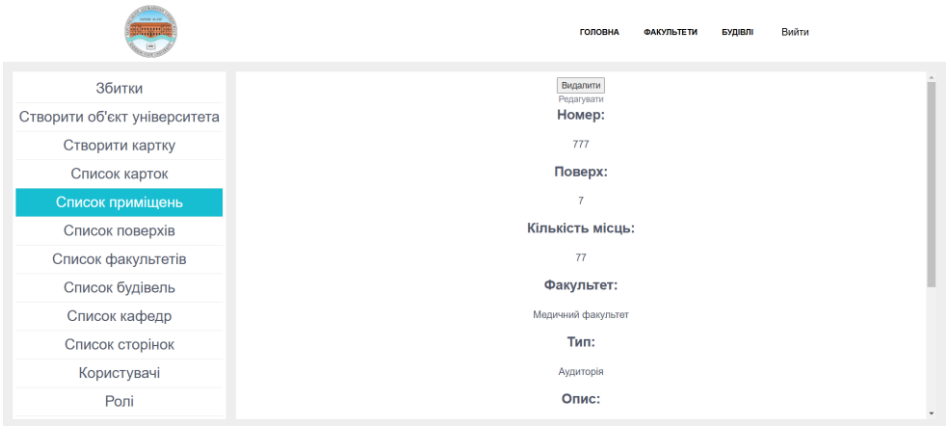

Рис. 3.5. Сторінка перегляду об’єкта.

3.12 Перенесення сторінки редагування об’єкту

Аналогічним чином до сторінки перегляду об’єкту було перенесено сторінки з редагування об’єктів з відредагованим для нового API запитом на редагування об’єкту. Після натискання кнопки «зберегти» відбувається попередня валідація полів, а після отримання відповіді від серверу – виводиться повідомлення про успішність виконання, або подробиці помилки.

3.13 Розробка функціоналу діаграм на сторінках поверхів

Минула версія сайту мала прототип діаграм, в які було заздалегідь вписано дані про поверх: кількість посадкових місць в приміщеннях та співвідношення типів приміщень. Цю систему було доопрацьовано автоматичним завантаженням та обробкою інформації з бази даних.

Для цього було додано бібліотеку ReCharts та створено скрипт, який робитиме запит до API для отримання інформації.

Перший тип діаграм – співвідношення типів приміщень на поверсі. Створюється новий масив з об’єктами, кожен з об’єктів містить два поля – назва типу приміщення та кількість приміщень даного типу. Після цього цикл проходиться по всьому масиву зі списком приміщень та підраховує кількість приміщень певного типу шляхом збільшення відповідних значень в новому масиві.

Другий тип діаграм відображає номери приміщень на вісі ординат, та кількість посадкових місць в цих кімнатах на вісі ординат. Для даної діаграми формується масив, який містить об’єкти з такою структурою: номер приміщення, кількість посадкових місць. Для формування даного масиву цикл проходить по масиву з отриманими даними та створює нові об’єкти, на основі оригінальних об’єктів.

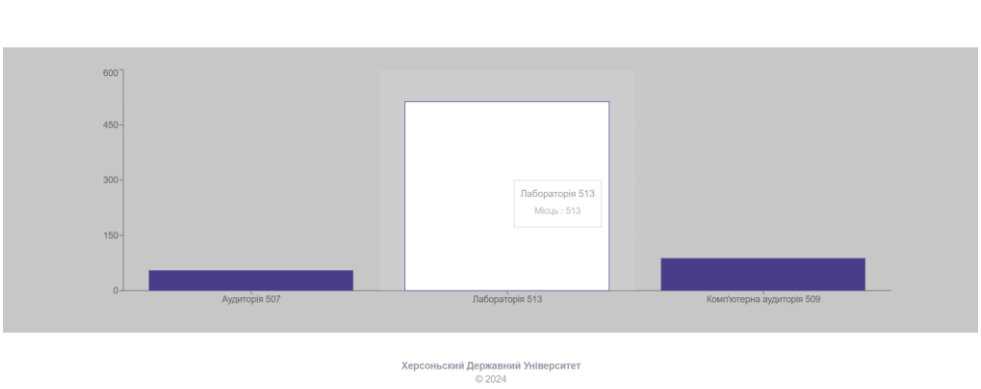

Рис. 3.6. Діаграма посадкових місць.

3.14 Завершення перенесення, тестування, виправлення багів

Після завершення перенесення минулої версії на React було протестовано кожну сторінку на баги, перед початком розробки нового функціоналу.

Першим знайденим багом стало завантаження зображень з директорії фронтенду, замість бекенду, так як минула версія мала відносний шлях в сусідню директорію, замість адреси бекенд серверу.

Також під час тестування було знайдено та виправлено дрібні помилки у верстці, серед них були як ті, які були на всіх пристроях, так і ті, які були лише на мобільних.

3.15 Можливість редагувати контент статичних сторінок з панелі адміністратора

Після завершення перенесення проекту було розпочато додавання нових функцій, в першу чергу було додано можливість редагувати статичні сторінки з панелі адміністратора.

Було створено ендпоїнт для головної сторінки, який надавав інформацію про зображення в слайді, заголовок, текст на кнопці та посилання.

Кожна сторінка має власний тип в MongoDB і зберігається в єдиному екземплярі.

Ендпоїнт завантаження підтримував завантаження файлів, для можливості замінити фонові зображення в каруселі.

На фронтенді було додано нову сторінку в панель адміністратора – список сторінок, яка мала аналогічний до інших списків функціонал.

До неї вручну було додано елемент «Головна», який має посилання на статичну адресу редагування головної.

Далі було створено окрему сторінку редагування головної, яка вміщає в себе 3 поля – Заголовок, посилання, множинне завантаження зображень, та кнопку «Зберегти». Після натискання цієї кнопки сторінка робить попередню валідацію та запит на редагування об’єкту головної сторінки. Після завершення операції на сервері користувачу повертається повідомлення про успішність або деталі помилки.

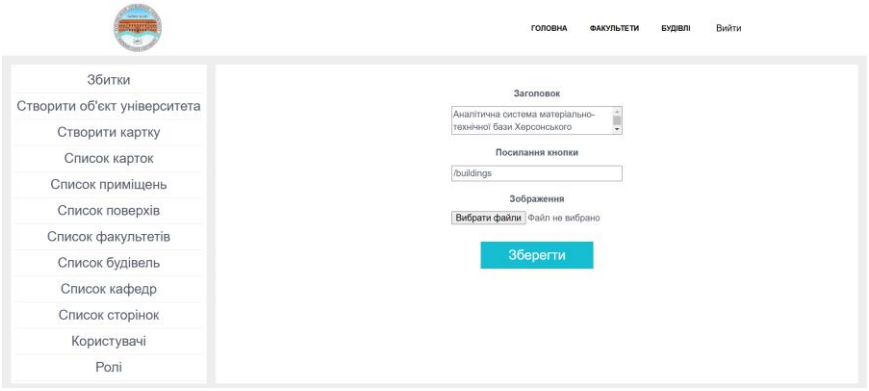

Рис. 3.7. Сторінка редагування головної

Далі до файлу головної сторінки було додано скрипт запиту інформації з серверу, який при успішному отримані даних додавав їх до сторінки за допомогою перетворення змінних React на DOM об’єкти.

Аналогічно до попереднього пункту було розроблено API сторінок будівель та факультетів, додано елементи «Будівлі» та «Факультети» в список зі сторінками та створено сторінки редагування полів.

Сторінки редагування містили наступні поля:

- Заголовок.
- Опис.
- Множинне завантаження зображень.

Аналогічно до попередньої сторінки було додано завантаження даних з серверу на оновлених сторінках.

3.16 Можливість створення інформаційних карток факультетів

Інформаційні картки на сторінці з інформацією про факультети, та на сторінках поверхів мали статичний контент і редагувались програмістом. Для полегшення роботи з ними було розроблено окрему сторінку в панелі адміністратора та API.

Спершу було створено повноцінний CRUD функціонал в API, який дозволятиме створювати об’єкти в базі даних з такими полями:

- Назва
- Іконка
- Кількість посадкових місць
- Колір
- Площа
- Кількість кафедр
- Кількість приміщень
- Бакалаврів денної форми
- Бакалаврів заочної форми
- Магістрів денної форми
- Магістрів заочної форми
- PHD денної форми
- PHD заочної форми

Далі було розроблено сторінку створення нових карток, вона має аналогічний функціонал до сторінки створення об’єктів, але має лише одну варіацію форми та надсилає запити на API створення карток.

Після цього було розроблено сторінку зі списком карток, яка робить запит на API та створює компоненти за кожним записом.

Далі було розроблено сторінку перегляду карток, вона має аналогічний функціонал та вигляд до сторінки перегляду об’єкту, з

додаванням поля «колір», який відображає текстове значення з фоном того кольору, який було задано.

Аналогічно було розроблено сторінку редагування, в якій можна редагувати кожне поле окрім назви, яка є ключовим словом.

Після завершення розробки частини адміністратора картки було імплементовано на сторінку з інформацією про факультети. Було створено компонент «картка», в яку передавався об’єкт картки з бази даних, та відповідно до значень полів створювався набір елементів картки.

Сторінка «факультети» робить запит на отримання всіх карток та в циклі послідовно створюється сітка зі згенерованих елементів.

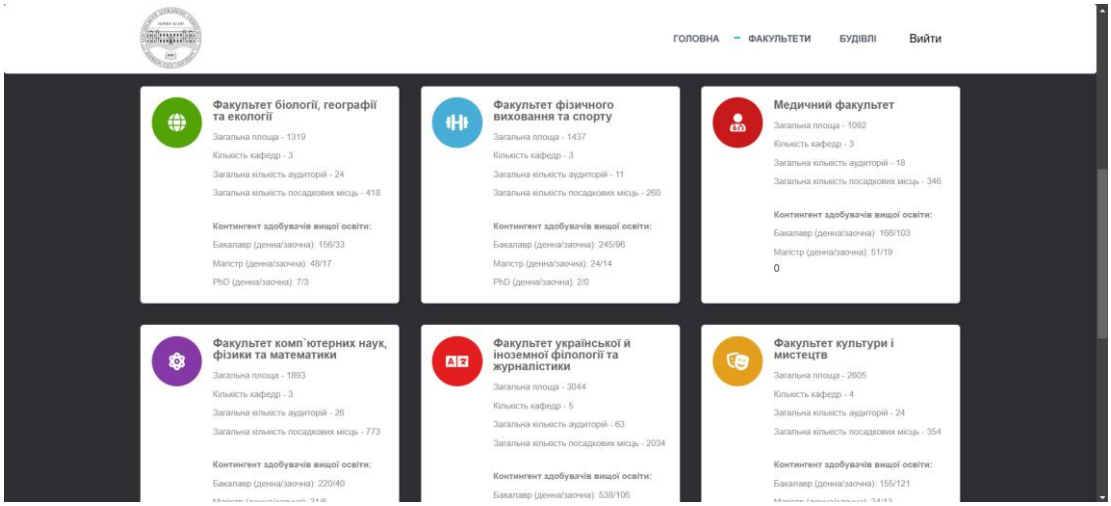

Рис. 3.8. Сторінка з інформаційними картками факультетів

Після цього було доопрацьовано API поверхів, при створенні та редагуванні яких з’явилась можливість обрати з випадаючого списку існуючу картку, щоб прикріпити до поверху. Вона відображається після схеми поверху.

3.17 Розроблення сторінки бази пошкоджень та збитків навчального закладу

Створення системи моніторингу збитків навчального закладу від обстрілів було почато з верстки фронтенд частини, для кращого розуміння того, що повинна мати бекенд частина.

В першу чергу було створено сторінку «збитки», яка дублювала функціонал сторінки зі списком об’єктів.

Далі до цієї сторінки було додано кнопку «створити запис про збитки», яка перенаправлятиме на сторінку аналогічну до сторінки створення об’єктів, але з полями для об’єкту збитків.

Після цього було створено два компоненти – текстове відображення загальної кількості збитків, яке відображало число з React стану, за замовчуванням – 0 грн, та випадаючий список з усіма будівлями разом зі збитками в обраній категорії. Список бере дані вже з існуючого API будівель. Значення про кількість збитків за факультетом підраховуватиметься в циклі, який буде проходитись по всім отриманим об’єктам з API при кожній зміні значення випадаючого списку. Кожне значення суми буде додаватись до react стану, який має значення 0 за замовчуванням. В подальшому цю систему можна вдосконалити додавши кешування значень, або прораховуючи їх на сервері при кожному оновленні списку збитків.

Також було розроблено прототип компонента з фільтрами, функціонал якого поки не було реалізовано до кінця.

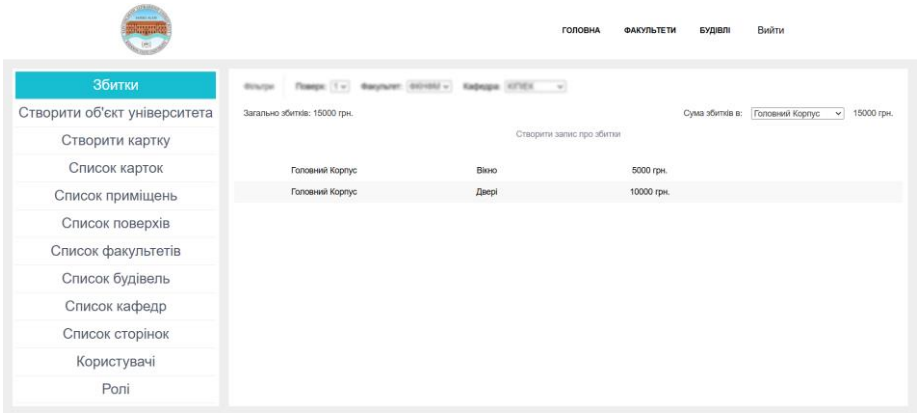

Рис 3.9. Сторінка зі списком збитків

Далі було розроблено сторінку створення запису про збитки. Вона містить наступні поля:

- Назва – текстове поле
- Будівля – випадаючий список, який генерується на основі списку всіх будівель, отриманого за запитом
- Місцезнаходження – текстове поле
- Стан – текстове поле
- Опис – текстове поле
- Сума – числове поле
- Множинне завантаження зображень

Для цієї сторінки було розроблено скрипт формування та відправки запиту на сервер при натисканні кнопки «створити», який надсилає JSON об’єкт, сформований зі значень всіх полів.

Далі було розпочато розробку CRUD функціоналу на бекенді, який дозволятиме створювати, читати, редагувати та видаляти об’єкти з описом збитків.

Після цього було розроблено сторінку перегляду запису про збитки, яка містила в собі відображення всіх полів, які можуть існувати в типі даного об’єкту в базі даних.

На основі сторінки створення нового запису було розроблено сторінку редагування, яка дозволяє змінити всі поля запису, окрім ключового поля ідентифікатора.

3.18 Розширення системи користувачів

Для допуску до редагування окремих типів об’єктів та окремих об’єктів було доопрацьовано систему користувачів, давши головному адміністратору можливість створення ролей та призначення їх користувачам.

В першу чергу було створено сторінку зі списком користувачів, яке звертається до API користувачів та виводить всіх за логіном. В майбутньому це API буде розширене полем «Роль», а до кожного елементу сторінки буде додано назву ролі.

Далі було створено сторінку створення нових користувачів, яка має наступні поля:

- Логін – текстове поле
- Ім’я – текстове поле
- Роль – випадаючий список, який буде формуватися з отриманих даних з API
- Пароль – текстове поле з прихованими символами
- Повторення пароля – текстове поле з прихованими символами

При натисканні на кнопку «створити» формується об’єкт та запит до API, яке в подальшому буде доопрацьоване можливістю додавати ролі до користувачів.

Далі було створено сторінку перегляду користувача, яка відображає всі поля, які містяться в базі даних, окрім поля «пароль».

Після цього було створено сторінку зі списком ролей, яка має аналогічний вигляд до сторінки зі списком користувачів та можливістю

створити ролі. Кожен елемент списку відображає назву ролі та чи є ця роль «суперадміном».

Далі було розроблено сторінку створення ролей, яка містить наступні поля:

- Назва – текстове поле
- Суперадмін – чекбокс
- Адмін - чекбокс
- Редактор - чекбокс
- Редактор збитків - чекбокс
- Будівля – випадаючий список, створений на основі отриманих даних з серверу
- Поверх - випадаючий список, створений на основі отриманих даних з серверу
- Факультет - випадаючий список, створений на основі отриманих даних з серверу
- Приміщення - випадаючий список, створений на основі отриманих даних з серверу

Кожна наступна роль, починаючи з «приміщення» і до «суперадмін» містить в собі привілеї попередньої ролі, та містить нові.

Привілея з назвою типу об’єкту дозволяє редагувати один конкретний об’єкт та всі залежні від нього об’єкти.

Після натискання на кнопку «створити» формується об’єкт ролі та формується та надсилається запит на створення ролі.

Далі було розроблено бекенд частину ролей – створено таблицю в базі даних, об’єкти в якій мають всі поля, які містились на сторінці створення ролі. До API було додано повний CRUD функціонал до таблиці.

Після цього в таблицю користувачів було додано поле «роль», яке містить в собі ідентифікатор ролі. Крім цього було додано випадаючий список «роль» на сторінці створення користувача, який генерувався динамічно на основі отриманої з серверу інформації, текстове відображення ролі в елементі списку користувачів та на сторінці перегляду користувача.

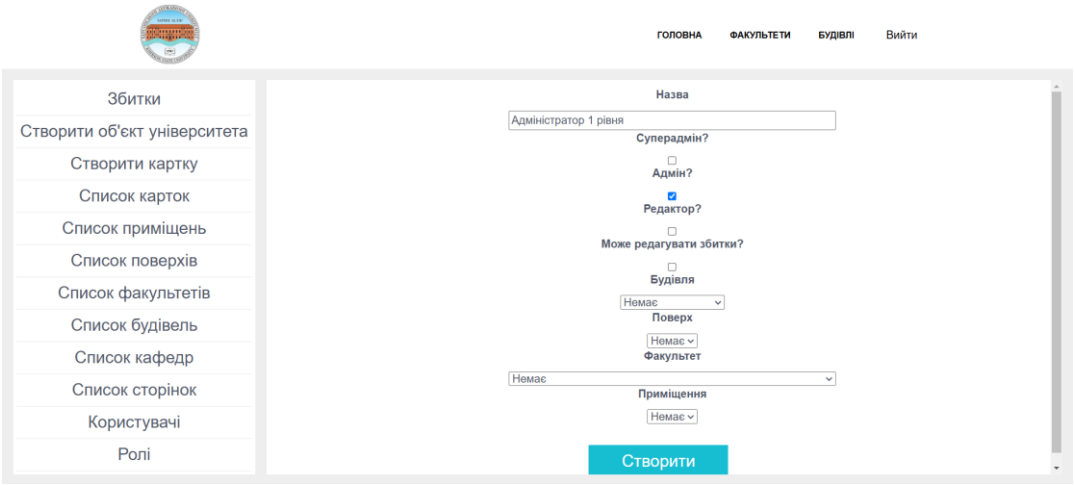

Рис 3.10. Сторінка створення ролі.

Далі було розроблено функції перевірок на доступність ендпоїнту певній ролі. Їх було додано до кожного ендпоїнту, який потребував доступ від певної ролі.

3.19 Доопрацювання сторінки приміщень

В минулій версії сторінка приміщень мала вигляд списку, в якій кожне значення поля об’єкту бази даних відображалось в окремому рядку. На сторінці не було заголовків, верстки та інтуїтивно зрозумілого розміщення об’єктів.

В новій версії сторінку було доопрацьовано, розділивши вивід даних в окремі групи – першочергова основна інформація справа зверху, карусель зображень зверху зліва, опис знизу зліва, а додаткова інформація та значення з датчиків – справа знизу.

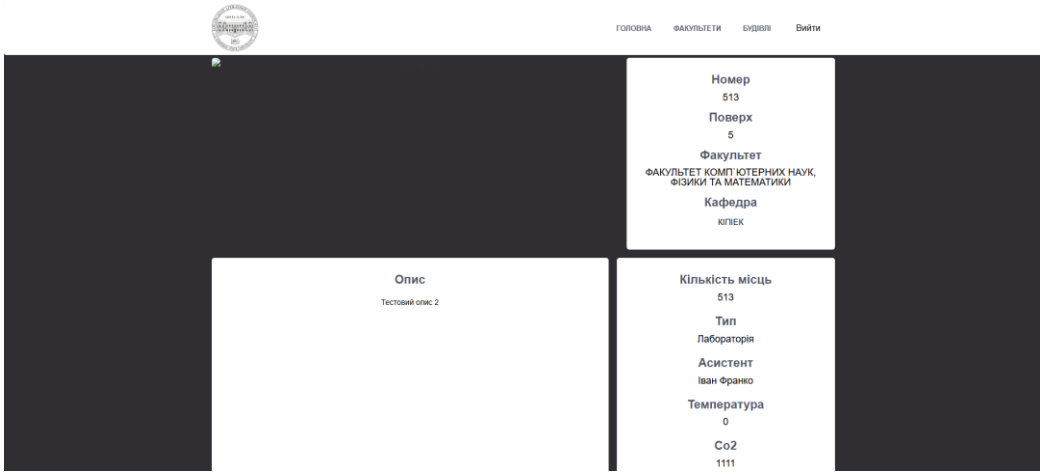

Рис 3.11. Сторінка приміщення.

ВИСНОВКИ

В роботі було проаналізовано відомі та релевантні системи, які мали елементи інтернету речей. На основі проаналізованих даних було розроблено вимоги до оновлення існуючої системи розумного навчального закладу.

В результаті проект зазнав змін у користувацькому інтерфейсі, які зробили використання системи більш зрозумілим для непідготовлених користувачів. Деякі сторінки було візуально покращено та приведено до єдиного стилю, що покращує досвід користувача.

В даному проекті було:

- Перенесено сайт зі стеку ExpressJs + Vue на React
- Покращено структуру проекту
- Створено розбиття повторюваних елементів на сайті на компоненти
- Виправлено баги старої версії
- Покращено дизайн сторінок
- Створено систему моніторингу збитків
- Створено систему інформаційних карток факультетів та кафедр
- Створено систему ролей користувачів
- Створено систему рівнів доступу до сторінок
- Розширено систему користувачів
- Додано можливість завантажувати SVG файли будівель та поверхів
- Розроблено динамічні інформаційні діаграми до сторінок поверхів
- Додано можливість редагувати контент статичних сторінок з панелі адміністратора
- Розроблено систему автоматичного заповнення сторінки з інформаційними картками факультетів

Перспективи розвитку проекту полягають у покращенні користувацького досвіду, створенню фільтрів та можливості сортувати елементи в панелі адміністратора, розширенню системи ролей.

Також в систему можна імплементувати функціонал збереження історії значень з датчиків та відображення її за допомогою діаграм.

В подальшому проект може бути розширений новими типами об’єктів, старі типи можуть бути доповнені новою інформацією, а SVG моделі можуть бути доповнені інтерактивними елементами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Войченко В. В. ІНФОРМАЦІЙНА СИСТЕМА МОНІТОРИНГУ СТАНУ БУДІВЕЛЬ ТА ПРИМІЩЕНЬ ЗАКЛАДУ ВИЩОЇ ОСВІТИ : стаття / В. В. Войченко наук. керівник к.пед.н. доц. М.О. Вінник ; Міністерство освіти і науки України ; Херсонський держ. ун-т, ф-т комп’ютерних наук, фізики та математики, кафедра комп’ютерних наук та програмної інженерії. – Херсон : ХДУ, 2024.
2. Войченко В. В. ПРОЕКТУВАННЯ ТА РОЗРОБКА СЕРВІСУ "SMARTUNIVERSITY" : кваліфікаційна робота / В. В. Войченко ; наук. керівник к.пед.н. доц. М.О. Вінник ; Міністерство освіти і науки України ; Херсонський держ. ун-т, ф-т комп’ютерних наук, фізики та математики, кафедра комп’ютерних наук та програмної інженерії. – Херсон : ХДУ, 2023.
3. Войченко В. В. Проектування Сервісу «Smart University» : курсова робота / В. В. Войченко ; наук. керівник к.пед.н. доц. М.О. Вінник ; Міністерство освіти і науки України ; Херсонський держ. ун-т, ф-т комп’ютерних наук, фізики та математики, кафедра комп’ютерних наук та програмної інженерії. – Херсон : ХДУ, 2022.
4. State of IoT 2024: Number of connected IoT devices growing 13% to 18.8 billion globally https://iot-analytics.com/number-connected-iot-devices/
5. CitiMan by Dhyan Networks and Technologies https://www.dhyan.com/solutions/smart-city-central-management-system/
6. Microsoft Azure https://azure.microsoft.com/
7. ThingsBoard website https://thingsboard.io/
8. ThingsBoard Cloud https://thingsboard.cloud/
9. Документація до NodeJs https://nodejs.org/en/docs
10. Документація до ExpressJs https://expressjs.com/

11. Документація Mongoose https://mongoosejs.com/docs/api/mongoose.html
12. Документація Multer https://www.npmjs.com/package/multer
13. Сторінка cookie-parser https://www.npmjs.com/package/cookie-parser
14. Сторінка bcrypt https://www.npmjs.com/package/bcrypt
15. Сторінка jsonwebtoken https://www.npmjs.com/package/jsonwebtoken
16. Сайт та документація до VueJs https://vuejs.org/
17. Сайт Bootstrap https://getbootstrap.com/
18. Сайт та документація до MongoDB https://www.mongodb.com/docs/
19. Документація до Postman https://learning.postman.com/docs/introduction/overview/
20. Документація до Ngrok https://ngrok.com/docs
21. Сайт Херсонського Державного Університету https://www.kspu.edu/default.aspx
22. Acuña, Leonardo & Narváez, Ray & Salas, Carlos & Magre, Luz & González, María. (2021). Smart UTB: An IoT Platform for Smart Campus. 10.1007/978-3-030-86702-7_21. https://www.researchgate.net/publication/354916569_Smart_UTB_An_IoT_Platform_for_Smart_Campus
23. Аналітична система матеріально-технічної бази Херсонського Державного Університету http://info.kspu.edu/
24. Документація React https://react.dev/learn
25. Документація Jest https://jestjs.io/docs/getting-started
26. Документація React Hook Form https://react-hook-form.com/docs
27. Документація React-router https://reactrouter.com/en/main
28. Документація React-cookie https://www.npmjs.com/package/react-cookie

29. Документація react-slideshow-image https://www.npmjs.com/package/react-slideshow-image

30. Документація Recharts https://recharts.org/en-US/api

31. Документація Yup https://www.npmjs.com/package/yup

32. Документація Vite https://vite.dev/guide/

33. Документація Express-session https://www.npmjs.com/package/express-session

34. Документація Gulp https://gulpjs.com/

35. Документація HTML https://developer.mozilla.org/en-US/docs/Web/HTML

36. Документація CSS https://developer.mozilla.org/en-US/docs/Web/CSS

37. Документація JavaScript https://developer.mozilla.org/en-US/docs/Web/JavaScript

38. Документація TypeScript https://www.typescriptlang.org/docs/

39. Самчинська, Я., & Вінник, М. (2013). ПРОСУВАННЯ Й РОЗПОВСЮДЖЕННЯ ПЕДАГОГІЧНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА РИНКУ УКРАЇНИ. Збірник наукових праць "Information Technologies in Education" (ITE), (15), 210–220. https://doi.org/10.14308/ite000409

40. Kasumu, Rebecca & Yinka, & Chidinma, Abe. (2024). The Role and Applications of Internet of Things (IoT) in higher education: Uses and ways IoT affects students' learning. International Journal of Multidisciplinary Research and Growth Evaluation. 5. 243-249. https://www.researchgate.net/publication/379178834_The_Role_and_

Applications_of_Internet_of_Things_IoT_in_higher_education_Uses_and_ways_IoT_affects_students'_learning