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

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Олександр Коваль

«___»_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра

спеціальності 121 «Інженерія програмного забезпечення»

освітня програма «Інженерія програмного забезпечення розподілених систем»

на тему: «Програмний модуль системи моніторингу споживання електричної енергії»

Виконав (-ла):
студент (-ка) IV курсу, групи ТІ-71
Коротич Максим Сергійович _____

Керівник:
доцент, к.т.н.
Ковальчук Артем Михайлович _____

Рецензент:
головний енергоменеджер КПІ ім. Ігоря Сікорського, к.т.н.
Шевченко Олена Миколаївна _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2021 року

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

спеціальності 121 «Інженерія програмного забезпечення»

освітня програма «Інженерія програмного забезпечення розподілених систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2021р.

ЗАВДАННЯ
на дипломну роботу студенту

_____
(прізвище, ім’я, по батькові)

1. Тема роботи _____
_____
_____

керівник роботи Ковальчук Артем Михайлович, к.т.н., доцент
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”____” травня 2021р. № _____

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи мова програмування python, javascript, typescript, dart вимоги до системи, технічне завдання, опис предметної області
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_____
_____
_____
_____
_____
_____
_____
_____

5. Перелік ілюстративного матеріалу

_____
_____
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”___”_____202___ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі	01.02.2021	
3.	Розробка архітектури та загальної структури системи	01.03.2021	
4.	Розробка структур окремих підсистем	01.04.2021	
5.	Програмна реалізація системи	15.04.2021	
6.	Оформлення пояснювальної записки	01.05.2021	
7.	Захист програмного продукту	12.05.2021	
8.	Передзахист	26.05.2021	
9.	Захист	16.06.2021	

Студент _____ _____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

У даній роботі розглядається процес аналізу вимог, проектування та розробки системи “Автоматичне робоче місце енергоменеджера”. Розроблений програмний продукт надає користувачу можливість легко вносити, редагувати та аналізувати дані пов’язані з документацією, вжитком енергоресурсів закладу, дотриманням температурних режимів тощо, у централізованому порядку, запобігаючи проблемам дублювання, втрати даних що є поширеними при обміну інформації засобами передачі текстових та табличних документів.

Система була надана на тестування зацікавленим особам, отримано відгук щодо бажаних/необхідних вдосконалень та виправлень, що можуть бути виконаними у наступних ітераціях розробки продукту.

Робота складається із: пояснювальної записки обсягом 62 сторінок що включає в себе 40 рисунків. Наявно 12 посилань на першоджерела та 2 додатки.

КАБІНЕТ ЕНЕРГОМЕНЕДЖЕРА, АНАЛІЗ ВИМОГ, РОЗРОБКА СПЕЦИФІКАЦІЇ, ТЕМПЕРАТУРНИЙ МОНІТОРИНГ, ОБЛІК ЛІЧИЛЬНИКІВ, БУДІВЛІ, КІМНАТИ, ДОКУМЕНТООБІГ

ABSTRACT

This paper considers the process of requirements analysis, design and development of the system "Automatic workplace of the energy manager". The developed software product allows the user to easily enter, edit and analyse data related to documentation, energy consumption, compliance with temperature regimes, etc., in a centralized manner, preventing duplication, data loss that are common during the information exchange via the means of text and spreadsheet documents.

The system was provided for testing to interested parties, feedback was received on the desired/necessary improvements and corrections that can be made in subsequent iterations of product development.

The work consists of: an explanatory note that is 62 pages long, which includes 40 figures. There are 12 links to sources and 2 appendices.

ENERGY MANAGER'S OFFICE, REQUIREMENTS ANALYSIS, SPECIFICATION DEVELOPMENT, TEMPERATURE MONITORING, METER ACCOUNTING, BUILDINGS, ROOMS, DOCUMENTS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АСКОЕ – автоматизована система обліку електричної енергії.

ЛУЗОД – локальне устаткування збору та обробки даних.

АСТОЕ - автоматизована система технічного обліку електроенергії.

API (англ. application programming interface) – прикладний програмний інтерфейс.

Frontend – термін, що загально позначає застосунки з графічним інтерфейсом користувача.

Backend – назва частини системи що відповідає за бізнес-логіку, взаємодію з даними, тощо.

ПЗ – програмне забезпечення.

JSON (англ. Javascript object notation) – це текстовий формат опису програмних сутностей, що переважно використовується при обміні інформації між комп’ютерами.

JWT (англ. JSON web token) – це стандарт передачі та збереження “ключа” доступу до застосунку у форматі JSON.

Token – термін, котрим називають ключ доступу.

Container/Контейнер – Відокремлене середовище роботи програмного застосунку, що є портативним та не залежить від впливу системи ззовні.

DB (англ. Database) – База даних, БД.

СУБД – Система керування базою даних.

Framework – це програмний каркас що складається з комплексу програмних рішень, що дозволяють значно полегшувати розробку системи. На відміну від окремих бібліотек є більш об’ємними.

HTTP (англ. Hypertext transport protocol) – протокол передачі текстової інформації мережею Інтернет.

ВСТУП

На великих підприємствах та організаціях, у сферу відповідальності котрих входить велика кількість корпусів, обладнання в них, лічильників тощо, існує проблема організації процесу збору та зберігання інформації пов’язаної з документацією, вжитком енергоресурсів.

У більшості випадків, відповідальний за організацію даних персонал веде облік з використанням звичайних текстових або табличних документів, що зберігаються на пристроях робітників. Цей підхід має багато недоліків, адже мають місце проблеми пов’язані з синхронізацією даних між пристроями відповідальних осіб, втратою документів у разі виходу з ладу робочого місця, неможливості внесення даних у оперативному порядку “на місці”, наприклад, коли відбувається введення даних показника лічильника, температури.

Системний комплекс “Кабінет енергоменеджера” націлений на полегшення роботи працівників служби енергоменеджменту. Продукт складається з трьох основних компонентів: мобільний застосунок, веб-застосунок адміністратора, та централізований сервер збору даних. Веб-застосунок використовується для повноцінної взаємодії з системою, внесенням статичних даних, документів, керування загальними налаштуваннями. Мобільний додаток, в свою чергу, націлений на використання оперативним персоналом, та дозволяє виконувати внесення/перегляд даних/показників “на місці”, з мобільного пристрою. Серверна частина виконує роль “центру”, що обслуговує користувачів додатків, та відповідає за збереження та роботою з даними, їх аналіз, агрегацію, експорт.

Не зважаючи на те що першочергово продукт був розроблений для використання саме у академічних закладах, елементи системи не зав’язані на окремій організації та її структурі, а тому можливе її використання установами з іншими напрямками роботи (наприклад, виробництва та офіси).

1. ЗАГАЛЬНА ХАРАКТЕРИСТИКА СИСТЕМИ

1.1. Мета роботи

Розроблене програмне забезпечення призначене для використанням енергоменеджментом організації (в даному випадку – університету) для зручного збору інформації про вжиток енергоресурсів, документаційну інформацію про характеристики споруд, обладнання тощо. Система повинна мати зручний графічний інтерфейс для взаємодію через ПК з використанням мережі Інтернет, та мобільний застосунок націлений на використання особами що виконують функції “збирачів” інформації.

Має бути підтримка збереження інформації про споруди, поверхи, кімнати, лічильники, стан лічильників, температурні умови. Повинно бути реалізоване введення та виведення інформації про ці сутності, можливість легкого експорту у загальновживаних форматах.

Система повинна мати підтримку декількох користувачів. Для цього пропонується реалізувати систему запрошень, за котрої користувач-адміністратор створює запрошення на створення облікового запису, а зацікавлена особа самостійно встановлює необхідну інформацію та описує свій профіль.

Більшість з описаних для системи бізнес-процесів та сутностей мають вузьку сферу використання. Через це актуальною є проблема обмеження доступу користувачам до зайвих розділів. Для обмеження функціоналу користувачів, програмний застосунок мусить реалізувати функцію прав доступу. За запропонованої системи, користувач може бути членом певної групи, а сама група має перелік прав до відповідних розділів. Групи впорядковуються в ієрархічній структурі для легкості їх організації.

1.2. Постановка задачі

Під час процесу розробки системи “Автоматизоване робоче місце енергоменеджера” необхідно виконати наступні задачі:

1. Провести аналіз вимог до системи, розглянути аналогічні рішення. Синтезувати кінцеві вимоги до продукту.
2. Розробити сценарії використання та алгоритми взаємодії користувачів з системою.
3. За отриманими даними розробити детальну специфікацію системи, виділити її прецеденти та сутності.
4. Провести розробку архітектури серверної частини застосунку.
5. Розробити дизайн користувацького застосунку, сформувати перелік та порядок екранів користувача.
6. Виконати розподіл задач між виконавцями.
7. Скласти план перевірки застосунку, провести тестування системи.
8. Розробити документацію для підтримки користувача.

Необхідно використати засоби розробки що дозволять максимально пришвидшити та полегшити процес розробки. З боку зацікавлених сторін не було висунуто вимог та обмежень щодо необхідних технологій, а тому, в рамках невеликої команди, найкращим рішенням в цьому плані є використання технологій що вже є добре знайомими для розробників, аби не витрачати час на їх вивчення.

Висновки до розділу 1

В першому розділі було розглянуто актуальність запропонованої роботи, визначено мету розробки системи, формалізовано та поставлено задачу на розробку програмного продукту, виділено кроки виконання роботи та виконано їх опис. Зазначена важливість використання зручних інструментів, що дозволяють значно зекономити час на розробку проекту.

2. ДОСЛІДЖЕННЯ ВИМОГ ДО СИСТЕМИ

Для вдалого аналізу предметної області, необхідно дослідити пов’язану з темою роботи літературу, проаналізувати поставлену задачу, знайти та порівняти вже існуючі суміжні та/або альтернативні рішення проблеми, дослідити досвід користувачів з уже існуючими системами та/або налагодженими мануальними процесами.

2.1. Існуюча система

Перед початком аналізу вимог до системи було проведено консультації з зацікавленою стороною. З її боку було надано:

- Описи існуючих бізнес-процесів.
- Приклади елементів/функціоналу зовнішніх систем що використовуються для роботи.
- Приклади робочих документів.
- Перелік побажань щодо функціоналу.
- Побажання щодо вигляду застосунку.
- Опис досвіду минулої команди розробки.
- Формальний опис цілей системи.

В результаті перемовин було виділено наступні нюанси щодо існуючих бізнес-процесів співробітників енергоменеджменту:

1. На даний момент часу не існує єдиної централізованої системи. Чітких меж у системи немає.
2. Деякі окремі зони відповідальності мають свої вузько направлені інструменти та програмні застосунки.
3. Окремі застосунки реалізують частину функціоналу автоматизованого збору та обліку технічної інформації.

4. Збереження інформації відбувається з використанням файлів та документів (рисунок 2.1). Кількість файлів є значною (рисунок 2.2).
5. Поширення даних відбувається вручну, передача на електронних носіях та/або електронною поштою, службою моментальних повідомлень.

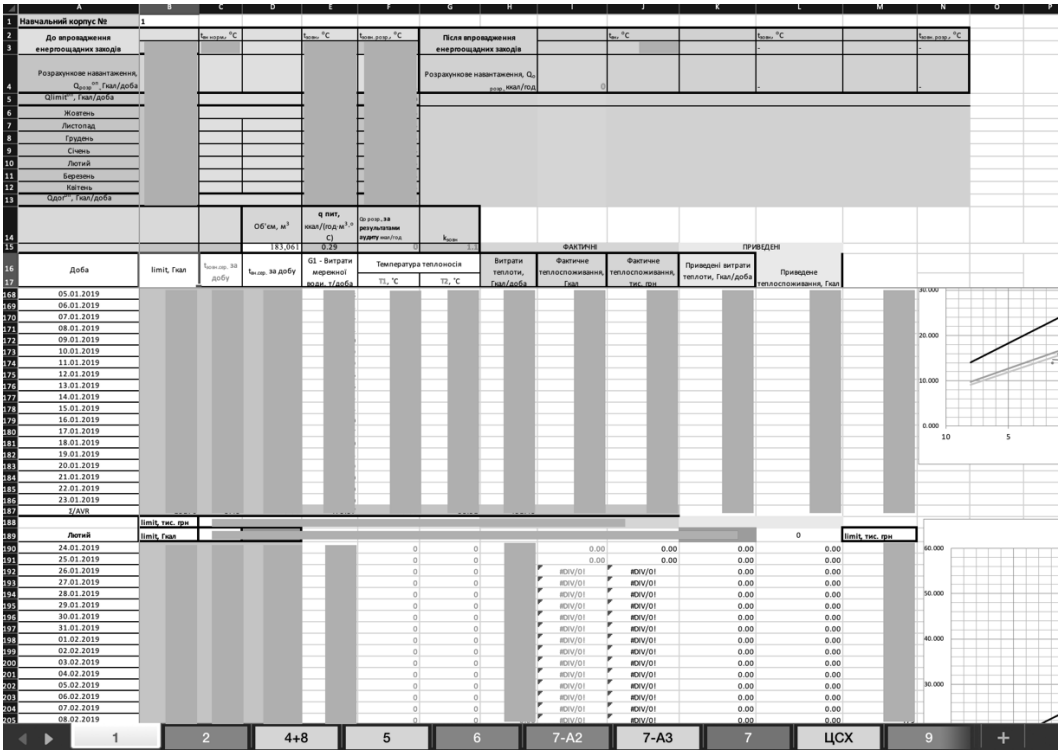


Рисунок 2.1 – Файл обліку вжитку енергоресурсів по корпусам (Інформація була прибрана задля збереження конфіденційності)

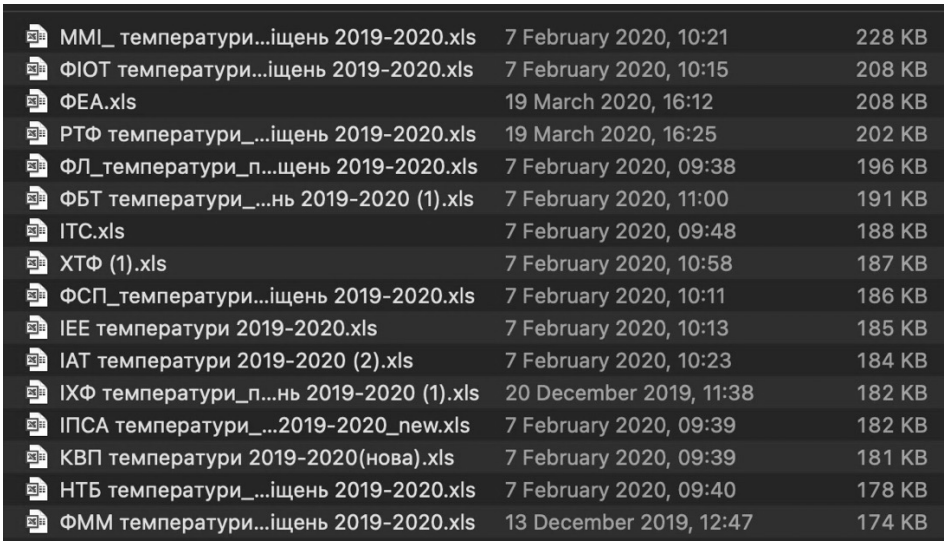


Рисунок 2.2 – Файли температурного контролю по корпусам

З урахуванням наданої інформації, були виділені наступні “больові” точки існуючих процесів:

1. Відсутність уніфікованої системи заважає легкому веденню інформації.
2. Використання великої кількості різних застосунків вимагає великих витрат на навчання, необхідна адаптація до кожного застосунку, є необхідність виконувати додаткові дії для перенесення інформації між застосунками та/або в форматі документів на ПК.
3. Збереження інформації у файлах призводить до наступних наслідків:
    a. Можливість втрати інформації в разі виходу робочої станції з ладу.
    b. Відсутність безпеки. Будь-який зловмисник що заволодів ПК, має можливість швидко вивантажити собі файли та видалити їх у користувача.
    c. Проблема у поширенні та версіюванні файлів. Великі архіви з даними необхідно постійно поширювати між робочими станціями. Це уможливлює втрату частин інформації при спільній роботі та завдає незручностей під час обміну даними.
    d. Файли займають багато місця на комп’ютерах, дублюються.
    e. Встановленні між файлами зв’язки призводять до проблем, якщо в кінцевого користувача відсутні окремі файли.
4. Відсутність системи контролю доступу. Користувач може отримати доступ лише до усього файлу, або не мати можливість вносити дані взагалі.
5. Невирішена проблема автоматизації збору інформації з лічильників та сенсорів (температури, вологості).
6. Необхідність своєчасної повірки і оновлення інформації.

Таким чином, централізована система котру було запропоновано розробити, націлена саме на вирішення цих гострих питань. Реалізація навіть частини запропонованого функціоналу зможе значно полегшити життя обслуговуючого (технічного) персоналу та адміністраторам.

2.2. Аналіз альтернативних рішень

Альтернативних систем, що повністю задовольняють поставленим вимогам, на даний момент не існує, або такі не були знайдені. Не зважаючи на це, було виявлено декілька готових продуктів, що частково реалізовують елементи функціоналу запланованої системи.

2.1.1. Система Jooby RDC & CMS

Jooby – це українська компанія з офісами в Україні та Швейцарії [2], що займається введенням у вжиток обладнання розумного освітлення, автоматизує процес зняття показників з лічильників електроенергії, газу, тепла, води тощо.

Компанія виконує розробку як і апаратних рішень – автоматизованих лічильників, сенсорів, розумного освітлення, опалення, так і програмних – Jooby RDC & CMS Dashboard.

Модуль Jooby CMS є універсальним рішенням для керування та аналізу енергоспоживання освітлення. Продукт надається у форматі серверного застосунку, до котрого під'єднуються розумні освітлювальні елементи, та виконується взаємодія з використанням WEB-інтерфейсу та/або мобільного застосунку під IOS та Android (рисунок 2.3).

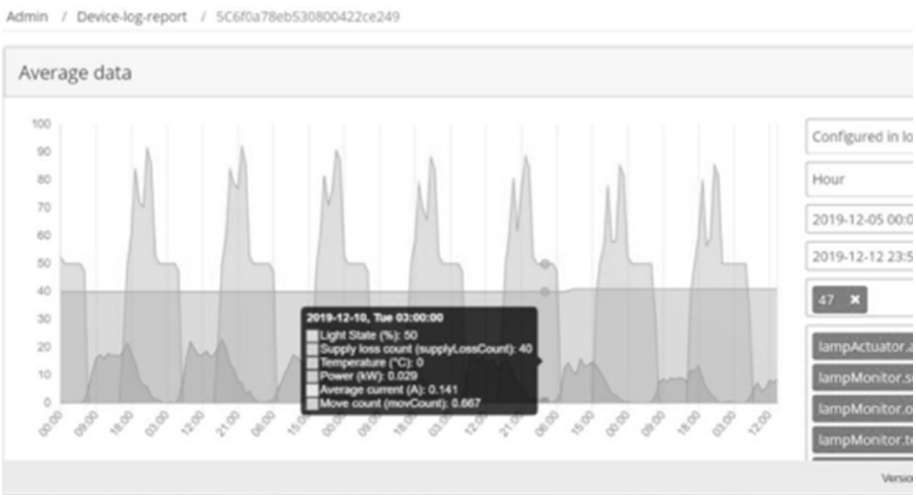


Рисунок 2.3 – Графічний інтерфейс застосунку Jooby CMS

Jooby CMS надає можливість налаштовувати наступні параметри системи освітлення:

- Яскравість освітлення в залежності від часу доби.
- Час роботи освітлення.
- Пов’язування з сенсорами: датчиками руху тощо.
- Групи діючих освітлювальних елементів.

Окрім цього, система надає детальний аналіз та звіт щодо часу і кількості енергоспоживання кожного освітлювального елементу що під’єднаний до неї. Необхідна інформація може буте переглянутою у форматі інтерактивних графіків, що оновлюються у режимі онлайн.

Модуль Jooby RDC Dashboard використовується для автоматизації процесу збору інформації з лічильників (електроенергії, води, тепла, газу). Система також постачається у форматі серверного застосунку з веб-інтерфейсом (рисунок 2.4).

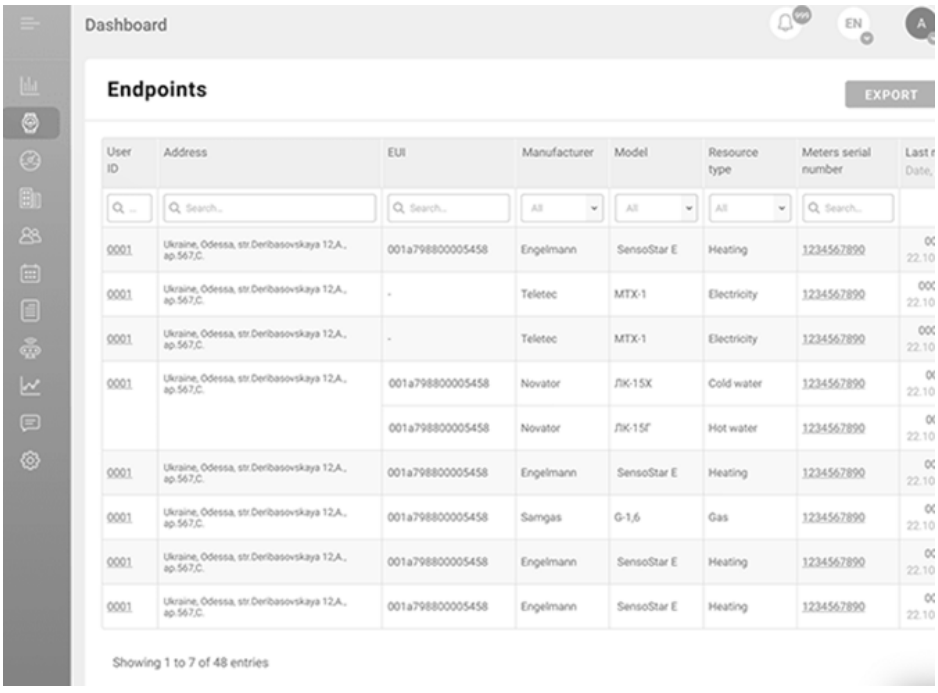


Рисунок 2.4 – Графічний інтерфейс застосунку Jooby RDC Dashboard

Користувацький інтерфейс надає користувачу можливість легкої взаємодії з можливістю перегляду інформації як і в табличній формі, так і у форматі онлайн-графіків.

Згідно з заявленими виробником характеристиками [2], системний комплекс Jooby має наступні переваги:

1. Швидкий доступ до стану приладів (лічильників та освітлення).
2. Зручний, легкий для використання інтерфейс, чіткий фокус застосунку.
3. Зручне створення документації, звітів. Експорт даних у форматі EXCEL.
4. Наявність журналу подій, що дозволяє зберігати сповіщення та повідомлення щодо помилок приладів та мережі.
5. Розумна аналітика даних, що дозволяє легше ухвалювати рішення на підставі наданої інформації.

Головним недоліком системи є те, що інтеграція “з коробки” можлива лише із пристроями виробництва тієї самої компанії. Для рішень з підтримкою різноманітних виробників необхідне замовлення інтеграції “під ключ”, що значно збільшує бюджет впровадження цієї системи до вжитку у вже існуючих організаціях. Окрім цього, це прив’язує замовників до виробника, що може призвести до проблем в разі збанкрутування останнього, або якщо розробку продукту буде завершено через його старіння.

Для задоволення вимог, поставлених до розроблюваної у дипломній роботі системи, продукту від компанії Jooby бракує:

- Облік будівель.
- Поверхи, кімнати.
- Обладнання кімнат.
- Моніторинг довкілля, температури.
- Документообіг.

Підводячи підсумки, продукт компанії є дійсно комплексним та завершеним, в сфері обліку інформації про вжиток енергоресурсів варто зважати на їх досвід. Але система не покриває усі поставлені у задачі вимоги, а тому не зменшує актуальність розроблюваного під час роботи програмного продукту.

2.1.2. Системи АСКОЕ, ЛУЗОД, АСТОЕ

Для автоматизації процесу збору інформації з електролічильників є поширеним використання систем АСКОЕ, ЛУЗОД та АСТОЕ [3] [4]. Вони складаються з наступних елементів (рисунок 2.5) [4]:

- Засоби вимірювання (Лічильники).
- Вузли збору даних (Центр керування).
- Сервери.
- Засоби зв’язку (GSM/Ethernet).

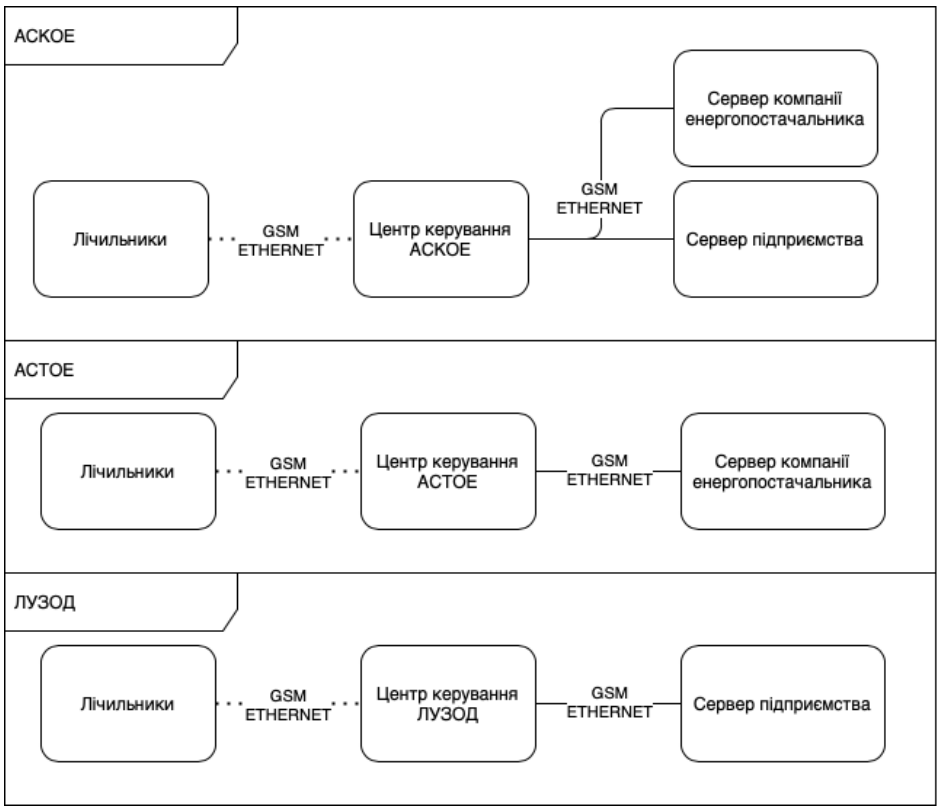


Рисунок 2.5 – Порівняння структури та ієрарії систем

Розглянувши взаємодію між компонентами системи більш детально, бачимо що лічильники (переважно із вбудованою наперед підтримкою системи), налаштовуються на відповідний сервер керування, з’єднуючись до них засобами Ethernet/GSM. Після цього, центр керування (вузол) починає виконувати збір інформації, та періодично або в режимі онлайн передає їх на сервери споживача та/або постачальника.

Основна відмінність наведених систем полягає у сторонах, що мають доступ до агрегованої інформації споживання. У випадку АСКОЕ – це як і постачальник, так і споживач, а у випадку ЛУЗОД та АСТОЕ – це відповідно лише споживач та постачальник [4]. Кожен підхід має свої переваги та недоліки. Наприклад, у разі інтеграції з постачальником, відсутня необхідність мануального звітування про споживання. Коли доступ є у споживача, то за ним залишається можливість виконувати власний аналіз щодо споживання енергоресурсів, та вести облік даних власноруч, наприклад, якщо цього потребує бухгалтерія. Система АСКОЕ є більш універсальною та зручною, адже включає в себе найкращі сторони обох систем. Причиною для обрання інших типів систем може слугувати обмеження, накладені політикою організації або самого постачальника.

До недоліку систем автоматичного збору належить можлива фрагментація в разі використання обладнання від декількох виробників, або якщо різні підрозділи однієї організації обслуговуються різними постачальниками, що накладають на ці системи власні обмеження та правила.

Варто зазначити, що було розглянуто системи збору даних лише з точки зору електроенергії, аналогічні (за структурою та суттю) рішення наявні і для інших видів енергоресурсів:

- Води.
- Газу.
- Теплоенергії.

Для уникнення повторень інформації описувати їх не будемо.

Резюмуючи, системи автоматичного збору інформації не можуть замінити розроблений у роботі продукт, але можливе їх паралельне використання для полегшення процесу збору та агрегації інформації в централізованій системі. Таким чином, дані отримані з АСКОЕ можливо вносити до загальної системи, та порівнювати з даними лічильників що були під’єднаними до зовсім інших систем, або взагалі уведено вручну.

2.3. Потреби користувачів

Після проведення аналізу отриманої інформації, її формалізації у вигляді більш зручного для узгодження в команді розробки, було виявлено наступні функціональні потреби користувачів у майбутній системі:

- Користувачі, система прав доступу.
- Документообіг (Тарифи, Договори, Документація).
- Споруди, поверхи, кімнати (Ведення документації: характеристики, відповідальні підрозділи, особи, додаткова інформація тощо).
- Вказання місця споруд на карті, перегляд місцевості на карті.
- Кімнати. Облік обладнання у кімнаті.
- Температурний моніторинг кімнат.
- Лічильники, дати повірок.
- Показники лічильників (Води/Газу/Тепла/Електроенергії).
- Нагадування про події.
- Вжиток енергоресурсів.
- Експортування/імпортування даних.

На рівні системи було вирішено закласти **на майбутнє** структурну та програмну реалізацію наступних пунктів:

- Інтерактивні карти, поверхові плани.
- Автоматичне звітування, розумний аналіз даних.
- Автоматичний збір даних з лічильників та АСКОЕ.

Підсумовуючи, система має великий набір функціоналу та передбачає розширення та доповнення його у майбутньому.

Висновки до розділу 2

У цьому розділі було розглянуто та досліджено проблематику предметної області системи. В цей час:

1. Було вивчено та проаналізовано вимоги користувачів до системи. Розглянуто наведені користувачами побажання, досліджено проблеми що виникали в користувачів під час планування та реалізації існуючих бізнес-процесів. Для виявлених проблем було запропоновано актуальні варіанти рішення.
2. Виконано аналіз існуючих програмних рішень. Розглянуто їх апаратні та програмні елементи. Зроблено детальний огляд архітектури та структури програмних частин наведених систем, визначено основні та другорядні особливості кожної з них. Для кожного із розглянутих застосунків було знайдено та класифіковано недоліки і переваги. Важливі особливості та сильні сторони було прийнято до уваги при розробці власної системи.

Як наслідок, синтезовано загальні вимоги до системи, визначено базовий функціонал системи, пріоритетність розробки кожного з елементів згідно до їх важливості в цілому.

3. СПЕЦИФІКАЦІЯ СИСТЕМИ

3.1. Функціонал користувачів та запрошень

Однією із вимог до системи є наявність можливості доступу до неї декількох користувачів. Для задоволення цієї проблеми система передбачає можливість створення та існування безлічі користувачів.

При першому запуску програмного комплексу, автоматично створюється головний користувач із наперед визначеними даними авторизації та групою із необхідним набором прав доступу. Головний адміністратор може використати ці дані для авторизації у мобільному та/або веб застосунку, в котрому подалі виконуватиме бізнес-процеси.

Система не передбачає можливості реєстрації за вимогою користувача (як це виконується на більшості загальнодоступних Інтернет-порталах), замість цього передбачається створення користувача та запрошення його до системи одним із її адміністраторів. Пропонуємо узагальнено розглянути цей процес на діаграмі (рисунок 3.1).

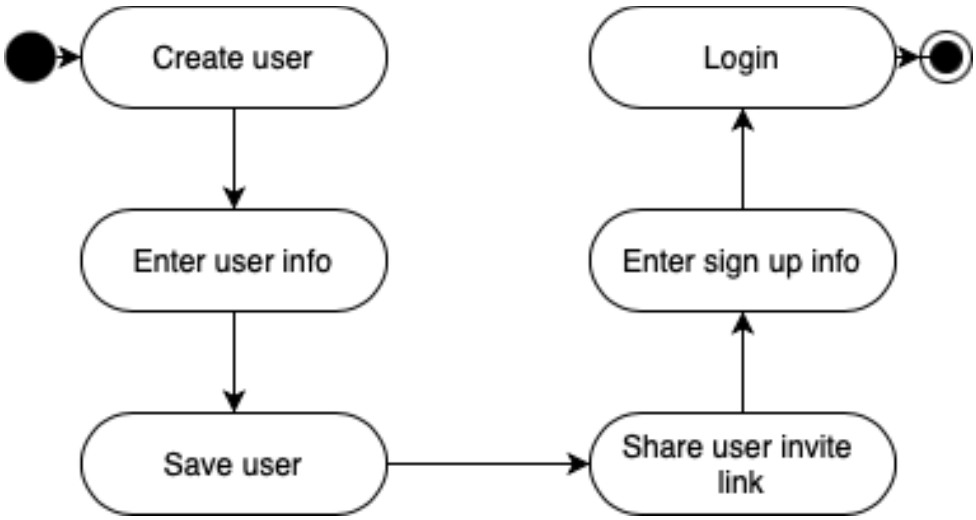


Рисунок 3.1 – Діаграма діяльності процесу створення користувача

Головний адміністратор має можливість створювати та видаляти користувачів. Для створення нового користувача, необхідно задати його поштову адресу, допустимо також вказання його ПІБ, фото профілю (що можна використовувати для диференціації користувачів в списках), та перелік контактів, що може буте переглянутий іншими особами в цілях звернення з приводу робочих причин.

По факту створення нового користувача застосунок автоматично виробляє запрошення на його долучення до системи. Запрошення являє собою одноразове та терміново обмежене посилання, перейшовши на котре новий користувач може виконати процес реєстрації у системі, увівши свій пароль та відредагувавши дані введені адміністратором. Для поширення посилання, адміністраторові системі необхідно скопіювати його з виведеного у інтерфейсі поля (рисунок 3.2), та передати його зацікавленій особі.

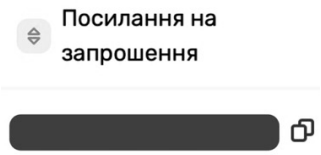


Рисунок 3.2 – Інтерфейс копіювання посилання на запрошення

По завершенню процесу реєстрації, користувач отримує доступ до системи. У власному профілі можливий перегляд та зміна загальної інформації, зміна паролю (за умови що користувач має старий), перегляд активних сесій користувача та видалення профілю у разі втрати актуальності, або зміні робочого місця користувачем.

Видалення профілю знищує усю приватну інформацію, але залишає в системі унікальний ідентифікатор користувача, що може бути використаним у разі якщо необхідно перевірити останні дії, зв’язки з сутностями тощо.

Відновлення доступу до системи без наявного паролю може бути виконаним через повторне створення запрошення на долучення до системи цього користувача.

3.2. Система груп та прав користувачів

При наявності доступу до системи у великої кількості осіб, виникає потреба у розділенні сфери їх взаємодії з нею, інколи обмеження доступу до окремих її елементів для покращення загального рівня безпеки. Саме для вирішення цієї проблеми, програмний продукт реалізовує систему груп та їх прав.

Користувач може бути членом однієї/жодної або декількох груп. Група має свою назву, перелік членів, окремі її члени можуть бути адміністраторами. Групи впорядковані у ієрархічному порядку (рисунок 3.3), тобто кожна група (окрім кореневої, що створюється по замовченню) має від нуля до декількох нащадків та одного родича.

Користувач-учасник групи отримує права, що проделеговані цій групі, та має можливість переглядати профілі інших членів цієї групи.

Користувач-адміністратор групи може додавати до неї користувачів, видаляти інших неадміністраторів з цієї самої групи, керувати правами та параметрами своєї групи, та усіма аспектами підлеглих до неї груп.

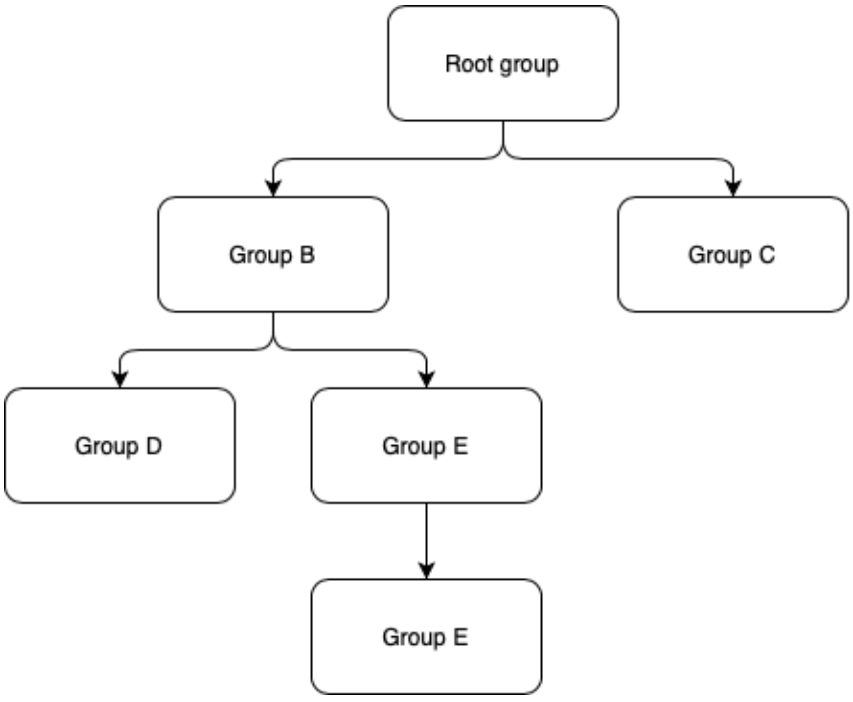


Рисунок 3.3 – Діаграма ієрархії груп

Кожна група має певний перелік прав що надаються її членам. Особливість системи прав полягає у наступних моментах:

- Доступна область прав підлеглої групи є завжди тотожною або меншою підмножиною прав батьківської групи (рисунок 3.4).
- Множина області доступних прав користувача є кон’юнкцією множин прав кожної з груп, до котрих він належить (рисунок 3.5).
- Користувач-адміністратор може змінювати множину прав підлеглих груп, але не своєї.
- Коренева група має повну множину прав, що не може бути зміненою.

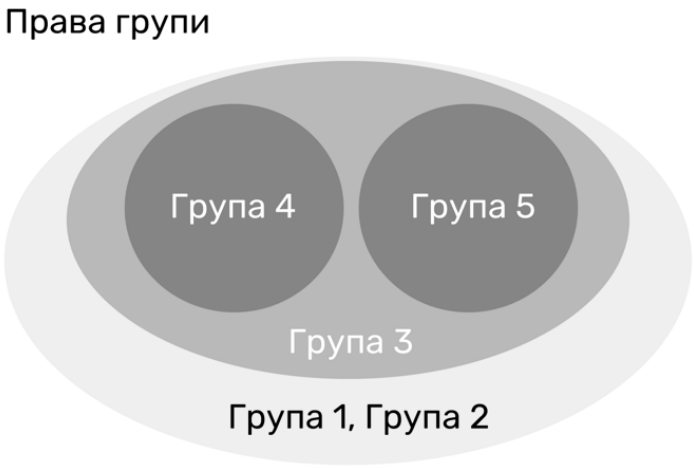


Рисунок 3.4 – Візуальне представлення області прав групи

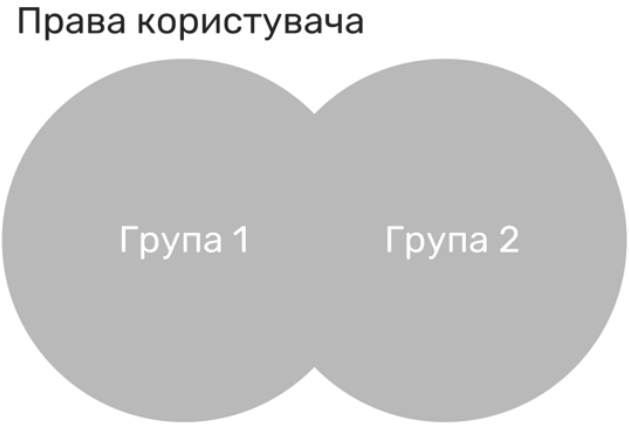


Рисунок 3.5 – Візуальне представлення області прав користувача

Кожна з незалежних сутностей що міститься у системі, має пару прав що відповідають за доступ та взаємодію з нею, а саме – право перегляду і окреме право на зміну/створення/видалення.

В системі виділені наступні групи прав користувача:

- Будівлі.
- Типи будівель.
- Поверхи.
- Кімнати.
- Обладнання.
- Показники середовища.
- Лічильники, їх показники.
- Користувачі, перегляд загального списку.
- Запрошення користувачів до системи.
- Групи, їх зміна.
- Документація.
- Тарифи.
- Договори постачання.

При наявності чи відсутності в користувача доступу до окремого модулю, графічний інтерфейс користувача виконує відповідні налаштування (наприклад, деактивує кнопки створення/редагування, кнопку доступу/детального перегляду розділу тощо) (рисунок 3.6).

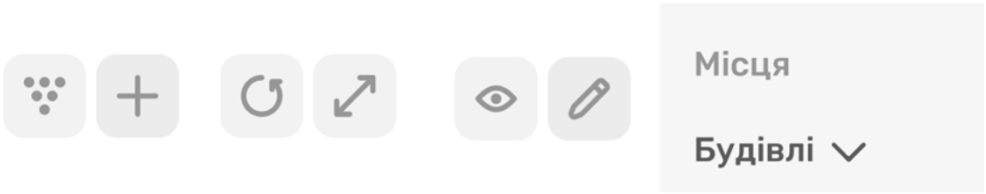


Рисунок 3.6 – Приклади компонентів інтерфейсу що стають неактивними/активними в разі наявності чи відсутності окремих прав

3.3. Авторизація користувачів

Для системи авторизації було вирішено використовувати новітній стандарт JWT – Json web token, що зберігає дані авторизованого користувача у вигляді підписаного JSON-об’єкта [5]. Перевірка вірності та відсутності підробки виконується через перевірку структури, підпису та обмеження строку дії, наявних прав користувача (рисунок 3.7):

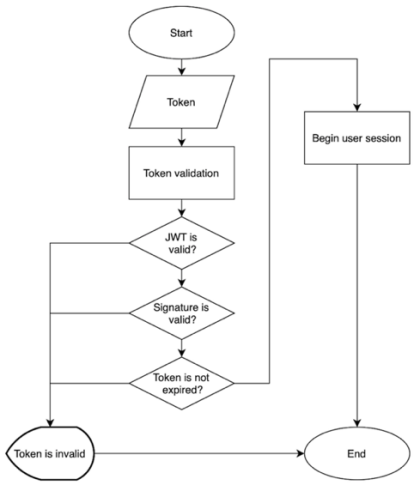


Рисунок 3.7 – Процес верифікації JWT-токену

Система авторизація на основі JWT працює таким чином, що при авторизації програмний застосунок користувача отримує пару refresh та access ключів, котрими він виконує доступ до серверу. Ключі підписані приватним ключем сервера та містять інформацію про користувача, а тому не має необхідності кожного разу виконувати доступ до БД для перевірки факту існування користувача, що є досить корисним у системах з розподіленими сервісами.

Ключ типу access має обмежений строк використання, це робить систему більш захищеною, адже навіть при його втраті, час користування зловмисником системи буде обмеженим. Для оновлення ключа access, використовується ключ refresh. В цей момент сервер робить перевірку на факт існування сесії, щоб не надати повторний доступ зловмисникам у разі її завершення.

3.4. Система фільтрування, сортування даних

У системі що розрахована на роботу з великими об’ємами однакових та/або різних даних, постає питання пошуку, сортування та фільтрування цієї інформації. Для забезпечення цього функціоналу було розроблено уніфікований стандарт та систему фільтрування і сортування даних.

При виконанні будь-якого запиту на множину даних, клієнт користувача може використати наступні параметри сортування (рисунок 3.8):

- Відсутній: сортування за порядком збереження в БД.
- Наявниий, з знаком плюс: Сортування за зростанням.
- Наявниий, з знаком мінус: Сортування за спаданням.

Порядок наданих до сортування полів також впливає на кінцевий результат, адже в такому разі буде виконано сортування по всіх полях почергово, з першого до останнього.

Формат параметру сортування (рисунок 3.9):

- Назви з декількох слів через нижній прочерк, малими літерами.
- (+/-). Вказує напрям сортування.
- Перелік полів задаються послідовно, через кому.

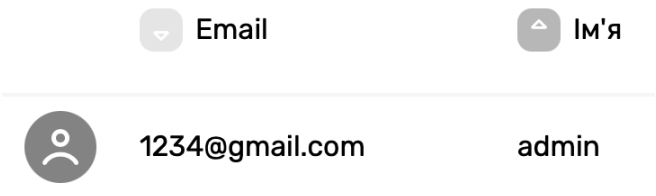


Рисунок 3.8 – Інтерфейс сортування

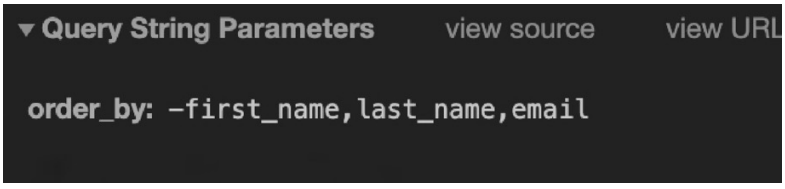


Рисунок 3.9 – Веб-запит з використанням сортування

При виконанні сортування можливе використання наступних параметрів (рисунок 3.10):

- __gt: Greater than – Більший за.
- __lt: Less than – Менший за.
- __gte: Greater equal than– Більший або рівний.
- __lte: Less equal than – Менший або рівний.
- __eq: Equal – Рівні.
- __neq: Not equal – Не рівні.
- __like: Like – Схожий на.

Формат параметру фільтрів є наступним:

- Назви з декількох слів через нижній прочерк, малими літерами.
- Параметр сортування, зазначається через подвійний нижній прочерк та назву параметру.
- Операнд. Число/текст за котрим виконується безпосереднє фільтрування.
- Перелік параметрів задається послідовно, через кому.

Окрім цього, параметри можна повторювати декілька разів. В такому разі між ними формується умова “АБО”. Це корисно, наприклад, коли необхідно знайти декілька сутностей маючи список їх ідентифікаторів, тощо.

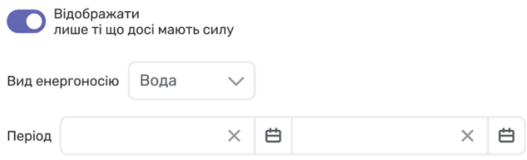


Рисунок 3.10 – Приклад інтерфейсу фільтрування даних

Для введення функціоналу сортування та фільтрування у вжиток без внесення великого об’єму змін до коду що відповідає за доступ до сутностей, було розроблено універсальні методи, що викликаються перед кожнім виконанням SQL-запиту в методах доступу до даних, та модифікують цей запит, додаючи до нього впорядкування з ORDER_BY та FILTER.

3.5. Розбиття даних на сторінки

Передача великих об’ємів даних по мережі Інтернет може викликати проблеми, адже існують наступні обмеження:

- Швидкість інтернету:
    - Обмеження постачальника.
    - Поганий зв’язок, навантажений канал.
- Обмеження кількості даних за тарфиом (Data cap).

Зважаючи на існуючі проблеми, є доцільним залучення функціоналу що дозволить завантажувати дані розумно, не використовуючи ресурси на те що не потрібно. Саме цього можна досягти через розбиття запитів дані на сторінки (також називається пагінацією, з англійської – pagination).

Система розбиття на сторінки даних працює для усіх сутностей, та може бути активована за надсилання у запиті параметрів (рисунок 3.11):

- page_size: Розмір однієї сторінки з даними.
- page_number: Номер сторінки.

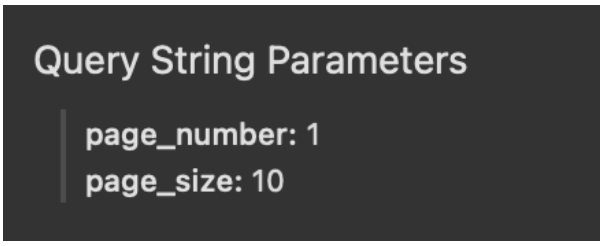


Рисунок 3.11 – Веб-запит з використанням сторінок

Таким чином, сервер виконує фільтрування, сортування даних, та повертає відповідь, обмежуючи її об’єм згідно з заданими параметрами сторінки. Відповідь сервера містить:

- total_size – Загальна кількість сутностей на сервері.
- items – Масив з переліком сутностей цієї сторінки.

3.6. Імпортування, експортування, агрегація даних

Окрім очевидних типів взаємодій з даними, таких як їх сортування та фільтрування, існує також потреба у її агрегації. Агрегація полягає у обробці даних, у зведенні їх до необхідного формату. До цього належить:

- Опис форматів вибірки:
    - Діапазон вибірки. Дата від і до.
    - Розмірність вибірки. Групування за проміжками часу:
        - Година.
        - День.
        - Тиждень.
    - Параметри вибірки:
        - Максимум/Мінімум.
        - Середнє арифметичне/геометричне.
        - Мода.
        - Медіана.
        - Перше/Останнє значення за період.
- Відновлення пропусків даних:
    - Інтерполяція:
        - Лінійна.
        - Поліноміальна.
    - Залишення пропусків.
- Поля що приймають участь в агрегації.
- Вибір формату постачання інформації:
    - Файловий:
        - Excel.
        - CSV.
    - Програмний:
        - JSON.
        - XML.

Для забезпечення цього функціоналу була розроблена специфікація системи агрегації даних. До вхідних параметрів алгоритму входять:

1. Сутність за котрою необхідно виконати агрегацію.
2. Перелік параметрів фільтрування, сортування.
3. Параметри розмірності вибірки.
4. Параметри відновлення пропусків даних.
5. Перелік, порядок полів що братимуть участь у агрегації даних.
6. Формат поверненої інформації.

Як результат, алгоритм повинен повертати список програмних сутностей, що конвертуються до необхідного формату та повертаються до користувачів. Таким чином можна виконувати як і експортування даних, так і отримання агрегованої вибірки по необхідним параметрам системи.

З точки зору користувача, агрегування даних зустрічається в двух сценаріях:

1. Експортування даних (рисунок 3.12).
2. Перегляд графіків. (рисунок 3.13).

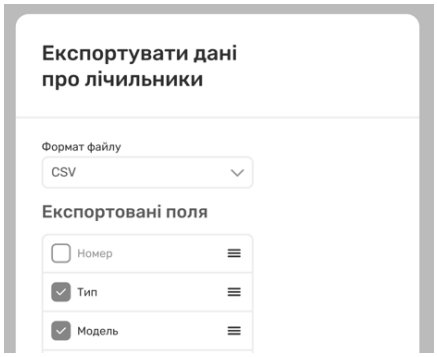


Рисунок 3.12 – Інтерфейс експортування даних

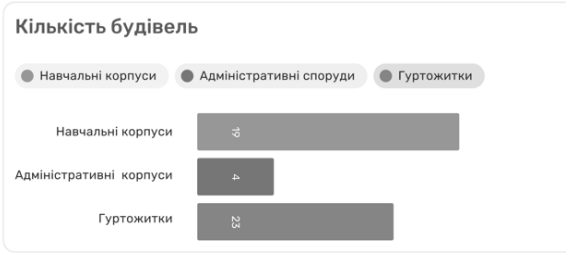


Рисунок 3.13 – Графік

Функціонал імпортування даних є другою важливою складовою частиною системи, що відповідає за агрегацію та збереження великих об’ємів даних. Імпортування даних з файлів дозволяє значно економити час, адже адміністратору не треба виконувати ручне перенесення даних вручну.

Під час розробки системи планується реалізація імпортування наступних типів таблиць:

- Корпуси.
- Лічильники.
- Кімнати та обладнання.

Через наявність великої кількості форматів файлів що вживалися, та складності у реалізації універсальної інтелектуальної системи імпортування даних з таблиць, було вирішено надати користувачеві можливість вибирати формат файлу власноруч, із наперед вказаних (рисунок 3.14).

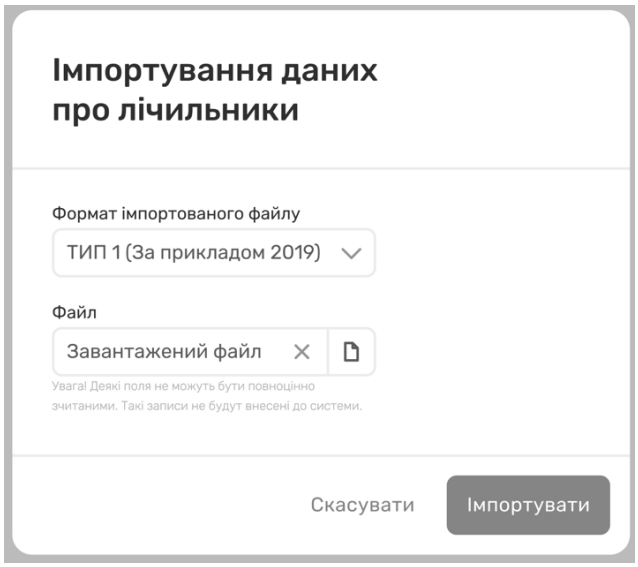


Рисунок 3.14 – Імпортування даних про лічильники

Таким чином, система імпортування даних надаватиме можливість виконувати швидке внесення даних для окремих сутностей. В разі виникнення потребі у імпорті даних іншого формату, можливе вдосконалення через додавання підтримки з боку серверного застосунку.

Висновки до розділу 3

У цьому розділі було описано програмну та структурну специфікацію основних програмних модулів системи. Детально:

1. Визначено сутність користувача, описано систему запрошень користувачів.
2. Запропоновано систему груп користувачів та наявних прав. Досліджено ієрархічну та структурну логіку груп, її характеристики. Було розглянуто систему прав, множини дії прав, вплив наявних прав на області доступу користувача і груп. Наведено необхідний графічний матеріал для роз’яснення концепції множин прав.
3. Обрано основу для технології системи авторизації. Розглянуто принцип роботи ключа доступу, етапи що виконуються сервером для перевірки достовірності токена.
4. Розроблено специфікацію для системи фільтрування і сортування даних на сервері. Запропоновано синтаксис взаємодії клієнтського застосунку з сервером, наведено приклад результату реалізації.
5. Розглянуто проблематику передачі великих об’ємів даних по мережі Інтернет, запропоновано використання концепції розбиття даних по сторінкам для запобіганню можливим проблемам.
6. Визначено поняття агрегації даних, зазначено важливість функціоналу агрегації, імпортування та експортування даних у системах з великими обсягами даних. Розроблено специфікацію та алгоритми роботи систем агрегації, імпорту, експорту даних. Наведено приклади використання цієї системи у кінцевому застосунку.

Як результат, було розроблено специфікацію основних функціональних особливостей системи.

4. ЗАСОБИ РОЗРОБКИ

Для розробки системи з великою кількістю функціоналу необхідне використання допоміжних технологій та засобів при розробці, адже вони надають значну перевагу у продуктивності праці та зменшують час на розробку.

4.1. Програмні інструменти розробки

4.1.1. Засоби проектування

Для процесу проектування системи та визначення вимог до неї було використано безкоштовний програмний застосунок “draw.io” (рисунок 4.1), що може бути вживаним із будь-якого актуального браузера в мережі Інтернет. Ця програма надає користувачу можливість швидко будувати діаграми будь-якого формату. Підтримуються графічні компоненти наступних специфікацій:

- UML, IDEF**.
- DFD, Entity relation.

У застосунку також наявна можливість додавання зображень, нотаток, довільних графічних елементів в разі потреби. Діаграми можна експортувати як і у загальнопоширених форматах (PDF, PNG), так і у пропрієтарному форматі для подальшої зміни.

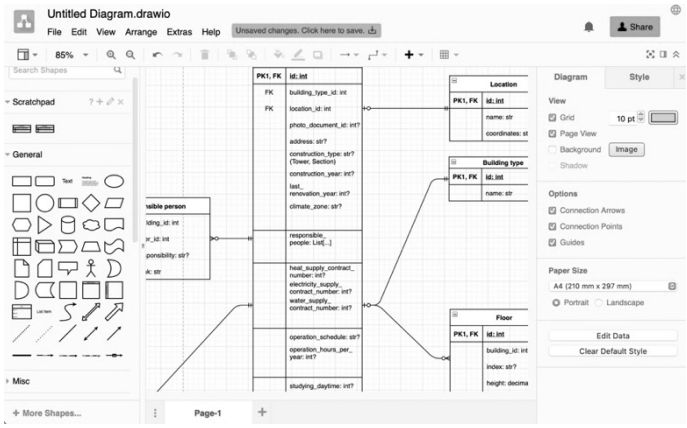


Рисунок 4.1 – Інтерфейс Draw.io

4.1.2. Макетування графічного інтерфейсу

Після визначення вимог до системи, паралельно із написанням бізнес-логіки, відбувається етап проектування інтерфейсу користувача. Цей етап є надважливим адже від якості виконаної роботи буде залежати зручність фінального продукту та як наслідок бажання клієнтів їм користуватися. Для проектування інтерфейсу застосунку був використаний застосунок Figma (рисунок 4.2). Він є безкоштовним для малих команд та поодиноких осіб, надає потужний інструмент для моделювання компонентів інтерфейсу, екранів тощо [9][8]. Наявна підтримка онлайн взаємодії, що дозволяє оперативно виконувати зміни та виправлення, та одразу надавати їх до осіб що відповідають за верстку.

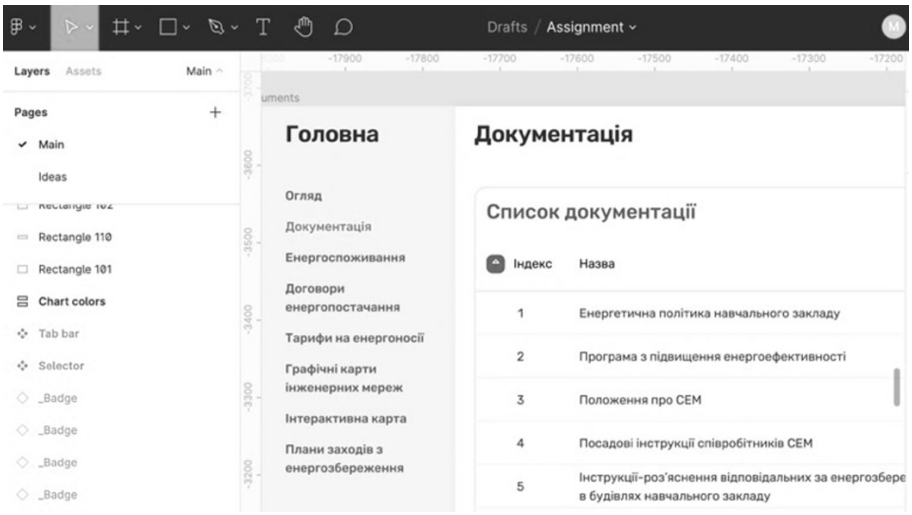


Рисунок 4.2 Застосунок Figma

4.1.3. Середовища розробки

Для виконання процесу безпосереднього написання коду застосунку, використовується набір IDE (З англійської - Integrated development environment) від компанії JetBrains (рисунок 4.3) [10], що надає зручний інтерфейс для написання програми. Застосунки поширюються на платній основі, але мають безкоштовні версії для академічного використання. Програма (рисунок 4.4) дозволяє користувачу виконувати швидкий пошук по коду, виконує перевірку на помилки, надає підказки при написанні. Окрім цього, вона має додатковий функціонал, такий як: інтеграція з GIT, вбудований менеджер БД тощо.

Рисунок 4.3 – Продукти JetBrains

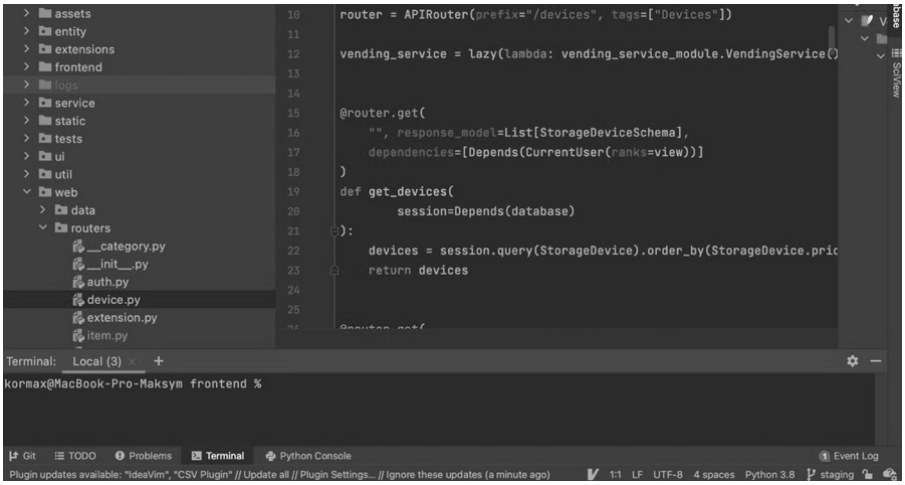


Рисунок 4.4 – Інтерфейс IDE

4.1.4. Засоби організації командної роботи

Під час активних етапів роботи, коли відбувалося паралельне розроблення декількох окремих програмних модулів, виникала проблема у впорядкуванні поставлених задач, відокремлення задач що блокують роботу інших розробників. Для полегшення процесу планування роботи було використано програмний інструмент від компанії Atlassian – Jira (рисунок 4.5).

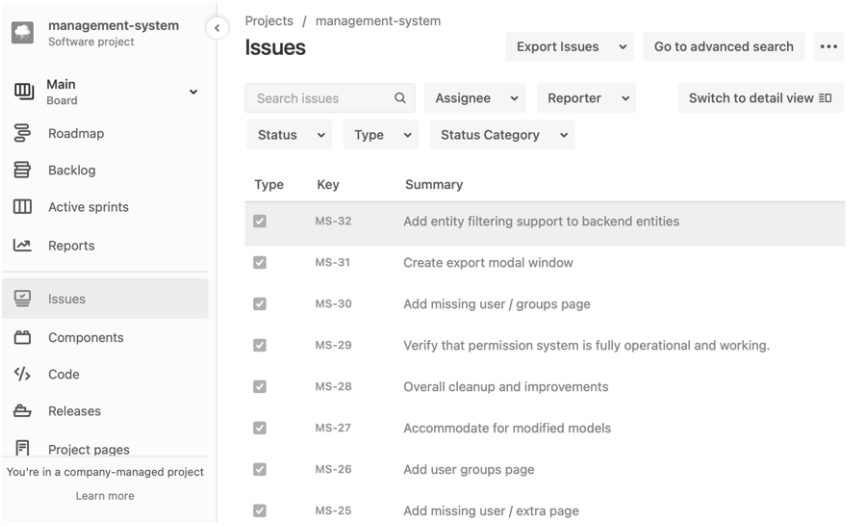


Рисунок 4.5 – Інтерфейс застосунку “Jira”

Інструментом, що дозволив додатково полегшити організацію командної роботи був засіб автоматичного створення документації – OpenAPI, та переглядачі документації з інтерактивним середовищем:

- Swagger (рисунок 4.6).
- Redoc (рисунок 4.7).

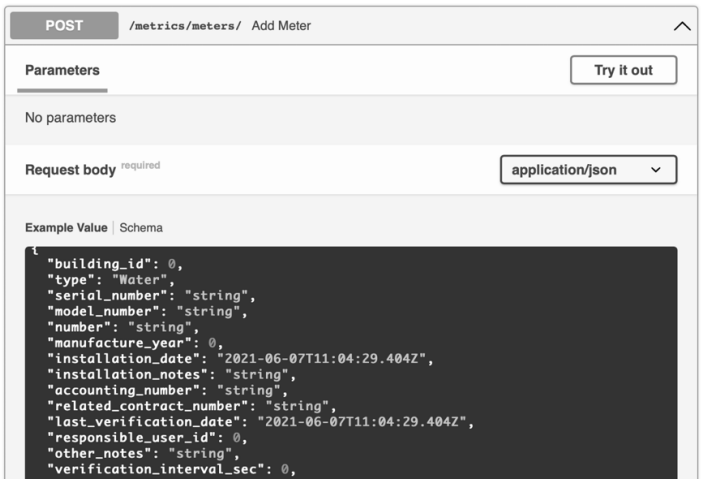


Рисунок 4.6 – Інтерфейс застосунку “Swagger”

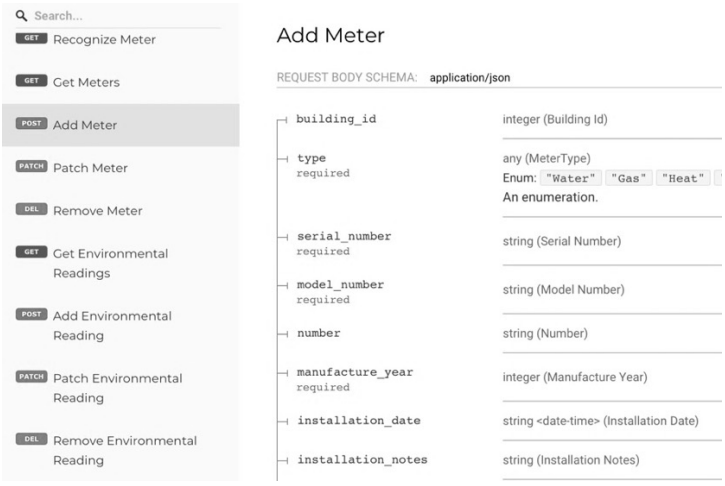


Рисунок 4.7 – Інтерфейс застосунку “Redoc”

Головною перевагою цих інструментів автодокументації є те, що вони формалізують та мінімізують комунікацію між розробниками backend та frontend частин застосунку, адже відпадає необхідність у проведенні консультацій однієї команди до іншої з приводу реалізації взаємодії з кожною новою сутністю системи.

4.2. Мови та технології розробки

Для розробки основної серверної частини застосунку, використовується мова Python. В залежності від сервісу, використовується фреймворк веб-серверу Django або FastApi. В кожного з них є окремі переваги:

- До переваг Django належить його велика популярність, наявність великої кількості функціоналу, супутніх модулів розширення.
- FastApi (разом із uvicorn) має вбудовану підтримку асинхронного програмування, автоматичної документації (рисунок 4.8) за стандартом OpenAPI для розробників застосунків що інтегруються з системою, інтегровану підтримку серіалізатору та десеріалізатору даних (із використанням pydantic)) [9].

Сервіс агрегації та аналізу даних був розроблений на мові Java. Саме ця технологія була обрана, адже для роботи з великою кількістю інформації є необхідним найоптимальніше використання системних ресурсів – процесора, пам’яті тощо (рисунок 4.8). Швидкодія та вимогливість є одним із основних недостатків мови Python. Для реалізації веб-серверу на мові Java використовується фреймворк Spring, для взаємодії з БД – Hibernate.

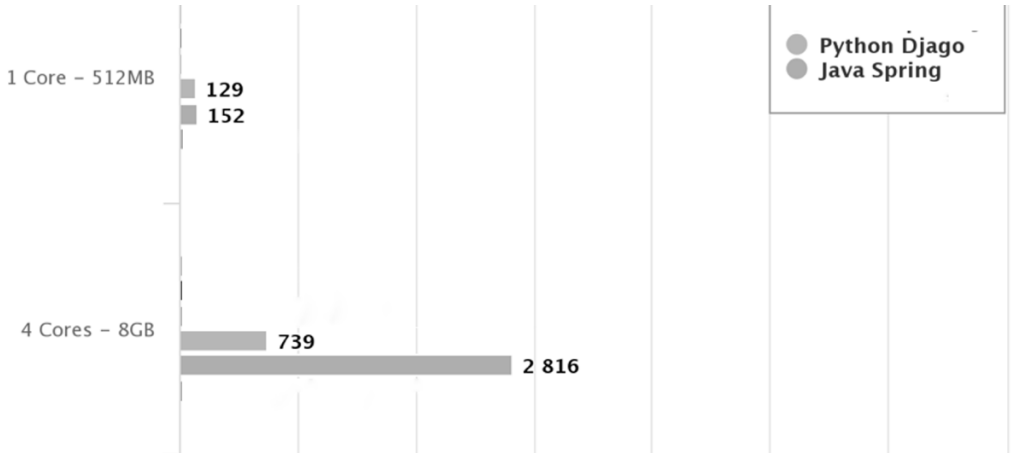


Рисунок 4.8 – Порівняння швидкодії (повернення даних з серверу)
в операціях на секунду в залежності від потужності серверу

Для розробки веб-застосунку використовуються мови TypeScript та Javascript + HTML/CSS. Основним фреймворком при розробці виступає Angular розроблений компанією Google [10]. Цей фреймворк дозволяє швидко розробляти веб-додатки з графічним інтерфейсом, використовуючи великий набір функціоналу із динамічного генерування сторінки та оновлення зображених даних. Додатково використовуються наступні інструменти:

- NGRX (Робота зі станом застосунку).
- Google maps (Відображення будівель на карті, обрання місця).
- Chart.js (Графіки) (рисунок 4.9).

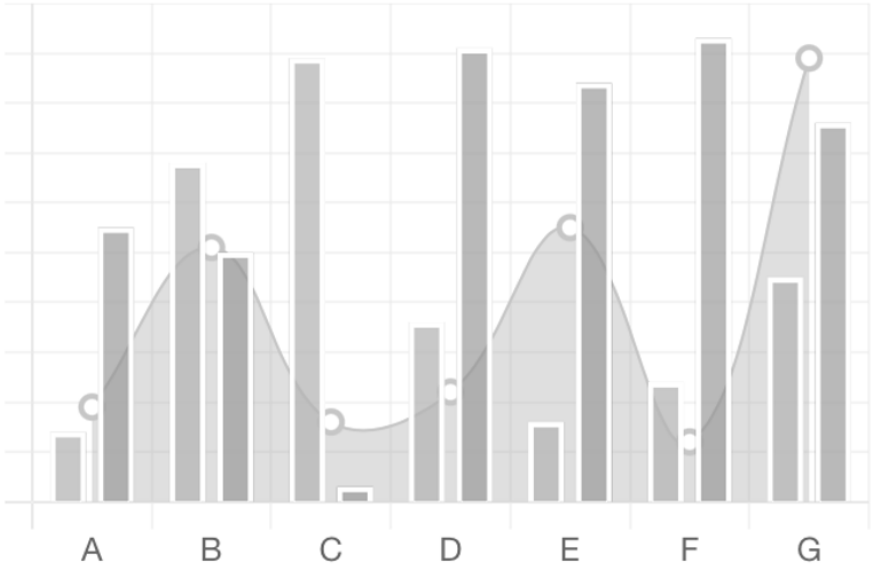


Рисунок 4.9 – Графіки Chart.js

Для розробки мобільного застосунку використовується мова Dart з використанням фреймворку Flutter, розробленому корпорацією Google [11]. Особливість та головна перевага цієї технології полягає у тому що розробник має можливість створити додаток одразу для декількох платформ, не пишучи жодної строчки коду що пов’язана лише з окремою платформою. Це дозволяє значно зекономити час розробки, та прибирає необхідність у вивченні специфіки кожної платформи під котру розроблюється застосунок.

4.3. Додаткові засоби використані при розробці

Для організації мікросервісної архітектури, була використана технологія контейнеризації, за котрої кожний компонент застосунку створюється та запускається у окремому віртуалізованому контейнері, що дозволяє їх легко створювати, організовувати, оновлювати, дублювати, заміняти та видаляти. На ринку існує декілька рішень, але для цього проекту було обрано Docker, адже він вважається найпопулярнішим, а тому має найкращу підтримку виробниками систем.

Для збереження даних сервісів, для кожного з них було використано контейнер іх БД PostgreSQL. До переваг цієї БД належать:

- Краща продуктивність у порівнянні із аналогами по типу MySQL.
- Підтримка нових типів даних:
    - Вкладені масиви.
    - JSON-структури.

Висновки до розділу 4

У цьому розділі було розглянуто застосовані під час розробки системи програмні продукти-застосунки та допоміжні інструменти розробки. Розглянуто загальну інформацію: тип, розробники, актуальність. Визначено мету та ціль їх використання, поставлені перед інструментами задачі. Описано ексклюзивний функціонал кожного з застосунків.

Як наслідок, проведено аналіз переваг кожного з інструментів, розглянуто перспективу використання цих інструментів в майбутніх проектах. описано залежні та підлеглі модулі, супутні застосунки.

5. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Розроблена система є комплексною та має велику кількість окремих та незалежних один від одного компонентів.

5.1. Загальний огляд системи

Розроблена система є комплексною та має велику кількість окремих та незалежних один від одного компонентів. Пропонуємо розглянути її загально, після чого провести опис кожного модулю більш детально. Для зручного уявлення системи, пропонуємо розглянути діаграму розгортання (рисунок 5.1):

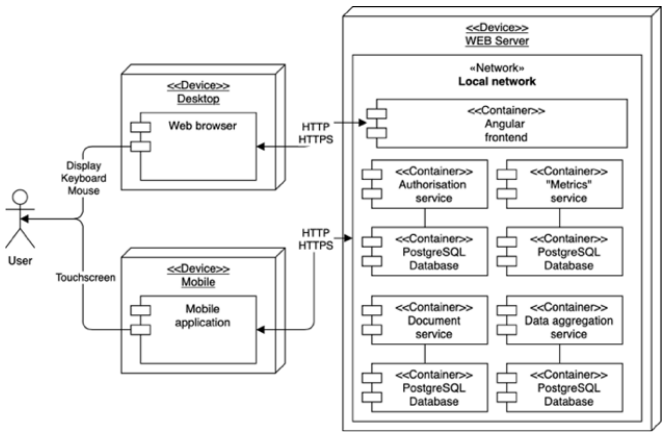


Рисунок 5.1 – Діаграма розгортання системи

Згідно з діаграмою, можна побачити що програмний комплекс можна поділити на дві основні групи:

1. Користувацька частина – мобільний та веб-застосунок.
2. Серверна частина – системні сервіси, їх СУБД.

Користувач взаємодіє лише з модулями користувацької частини, в той час як ці застосунки взаємодіють з модулями backend-системи через HTTP/HTTPS. Кожен з елементів цих груп було розроблений окремо, та їх програмна реалізація значно відрізняється. Розглянемо їх більш детально у наступних розділах.

5.2. Сервіс авторизації

До сфери відповідальності сервісу авторизації належать наступні прецеденти (рисунок 5.2):

Рисунок 5.2 – Прецеденти сервісу авторизації

Узагальнюючи, можна виділити наступні групи прецедентів:

- Авторизація/Вхід/Вихід.
- Реєстрація за запрошенням.
- Керування користувачами.
- Керування групами, їх членами, правами.
- Керування налаштуваннями безпеки:
    - Перегляд та керування сесіями.
    - Зміна паролю.

Для реалізації цих прецедентів, були виявлені та розроблені необхідні програмні сутності [1][7]:

- Користувач (Ім’я користувача, прізвище, ім’я, фото профілю, опис, флаг факту активації, флаг факту видалення).
- Контактні дані користувача.
- Група користувачів (Назва, флаг факту видалення, група-володар), користувач-група (Наявність права адміністрування, користувач, група).
- Права доступу.
- Запрошення.
- Дані авторизації (Реалізовано лише пароль, можливе розширення для альтернативних методів у майбутньому).
- Сесія (Дата завершення, інформація про пристрій користувача, дата останньої активності).

Для представлення зв’язків між сутностями, була побудована модель зв’язків сутностей (рисунок 5.3):

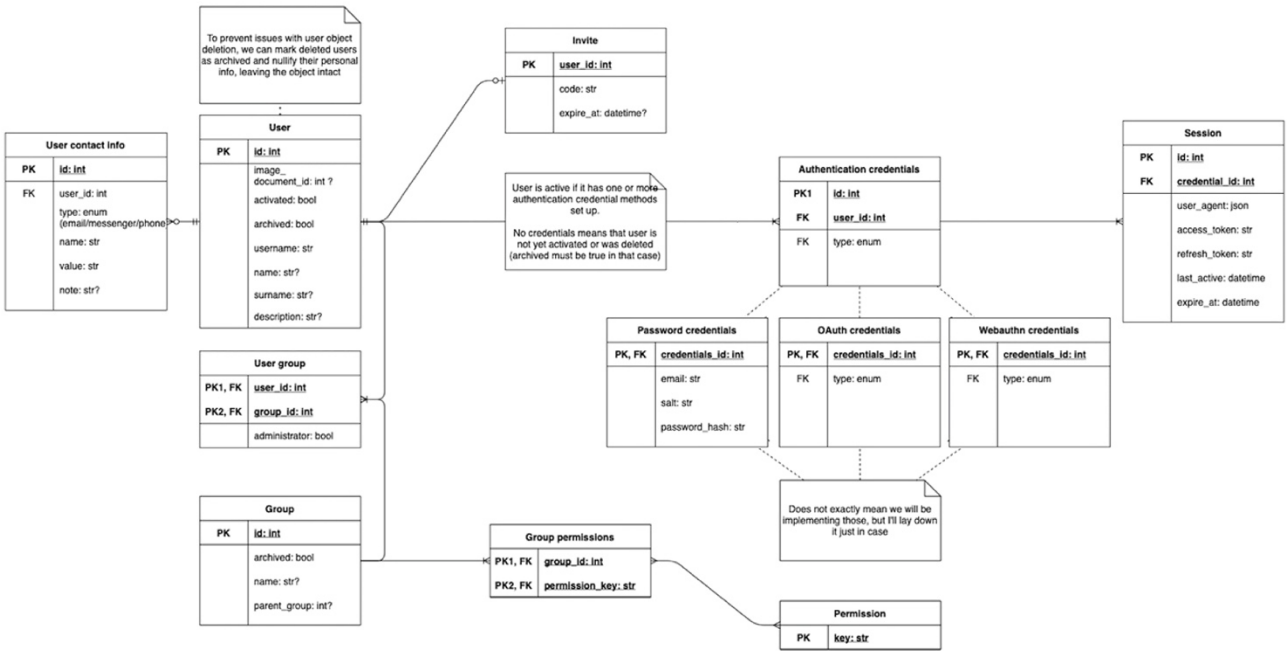


Рисунок 5.3 – Діаграма зв’язків сутностей

З програмної точки зору, сервіс авторизації було реалізовано на мові Python 3 із використанням наступних технологій:

- Django Framework – Ядро. Фреймворк побудови веб-застосунків [11].
- Django REST Framework – Для побудови REST-API.
- Django ORM – Для створення програмних сутностей та зв’язку їх із базою даних, міграцією даних при зміні специфікації [11].
- Django Auth – Для організації логіки авторизації.
- Django Test – Для тестування.
- Psycopg – Драйвер взаємодії з базою даних.
- Python-dateutil – Бібліотеки взаємодії з датами та форматами дат.
- Pyyaml – Бібліотека роботи з YAML-файлами.
- Uritemplate – Бібліотека що використовується як допоміжний інструмент створення документації у форматі OpenAPI.

Програмний код сервісу поділений на наступні модулі:

- main – Базові налаштування застосунку.
- model – Програмні сутності.
- views – Методи API.
- settings – Загальні налаштування застосунку.
- wsgi – Налаштування веб-серверу.
- urls – Зв’язок методів із шляхами URL.
- serializers – Методи серіалізації/десеріалізації даних.
- utils – допоміжні методи (Генерація пробних даних, парсинг даних тощо).

Сервіс авторизації відповідає за створення JWT-токена, по котрому користувач вже виконує доступ до захищених методів як і сервісу авторизації, так і інших сервісів системи. Ключ підпису токену знаходиться на сервері авторизації, та зберігається у безпечному місці.

5.3. Сервіс документів

Основною метою використання сервісу документів є збереження та агрегація усіх використаних у системі файлів, ведення документообігу, збереження договорів/документації, тощо (рисунок 5.4).

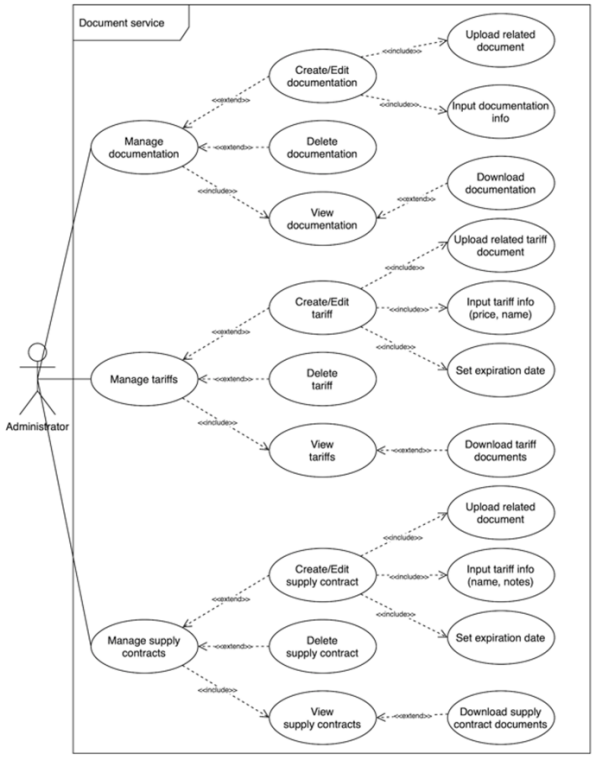


Рисунок 5.4 – Прецеденти сервісу документів

Наведені прецеденти можна винести до трьох загальних груп:

1. Документація.
2. Тарифи на енергоносії.
3. Договори постачання енергоносіїв.

Для кожної з наведених груп є спільними наступні прецеденти:

- Перегляд:
    - Завантажування файлів.
- Створення/Редагування:
    - Підвантажування файлів.
    - Введення загальної інформації.

Для реалізації всіх необхідних прецедентів були розроблені відповідні моделі. До сутностей цього сервісу належать (рисунок 5.5):

- Документ, фактично – будь-який файл. Має свій тип, назву, розмір, шлях до доступу на сервері.
- Документація – це файл інформаційного характеру що енергоменеджер зберігає на сервер для перегляду відповідальним персоналом.
- Договір постачання – документ про договір постачання, дата початку та завершення, номер, тип договору, посилання на договір/файл договору.
- Тариф – відповідний указ, тип, ціни для кожної категорії споживачів, додані до нього нотатки.

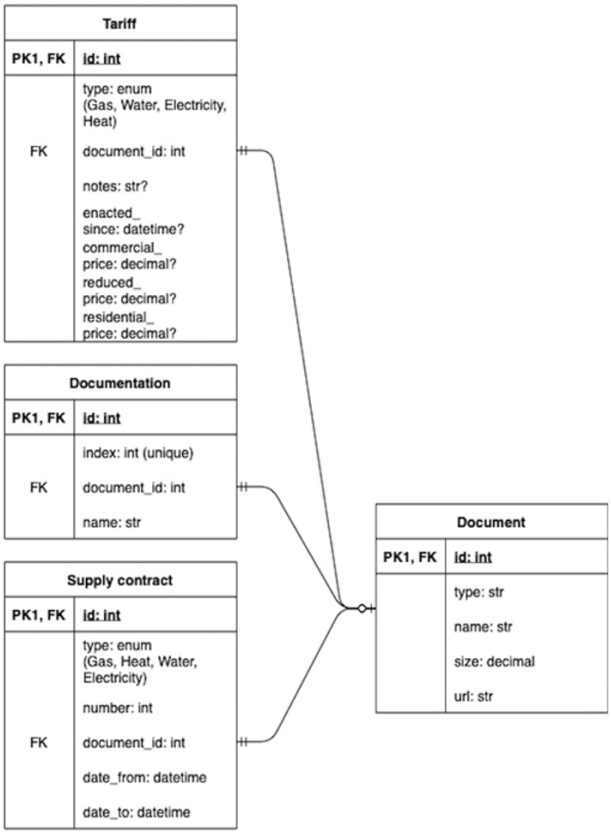


Рисунок 5.5 – Діаграма зв’язку сутностей сервісу документів

Стек програмних технологій сервісу документів є повністю ідентичним до такого у сервісу авторизації (Python, Django, Django ORM, *). Структура проекту та коду ідентична, але включає відповідно інші програмні сутності та методи, що були зазначені попередньо. Уникнемо повторень.

5.4. Сервіс “Метрики”

Сервіс “Метрики” містить левову частку усього функціоналу системи. До сфери його відповідальності належать (рисунок 5.6):

- Місця (Координати на карті, назва).
- Тип будівлі (Довільний, використовується для угрупування даних).
- Будівлі (Назва, загальні характеристики, номер, місце).
- Поверхи (Будівля, висота, номер).
- Елементи поверхового плану (Місце на плані).
- Кімнати (Площа, номер, відповідальний підрозділ, поверх).
- Обладнання кімнат (Вікна, освітлення, опалення тощо).
- Знімок даних про середовище у кімнаті (Температура, вологість).
- Лічильники (Серійний номер, тип, будівля встановлення, нотатки).
- Знімок даних з лічильника (Тип, лічильник, показники, додаткові параметри пов’язані з типом лічильника).

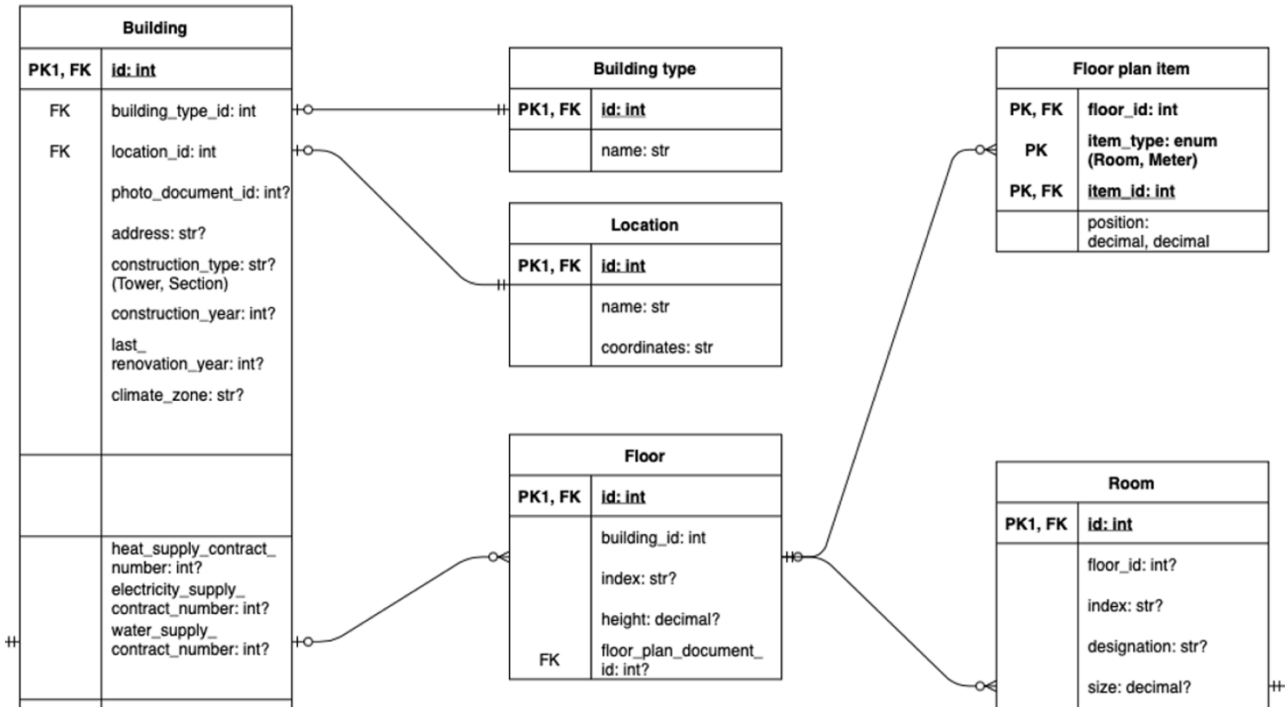


Рисунок 5.6 – Фрагмент діаграми зв’язку сутностей сервісу “метрики”

Сервіс реалізовано мовою Python3 із використанням наступного стеку технологій:

- FastAPI – Фреймворк зручної побудови веб-серверів із вбудованим функціоналом створення документації [12].
- uvicorn – Веб-сервер що використовується фреймворком. Є більш швидкодіючим аніж популярні аналоги.
- Pydantic – Бібліотека для опису сутностей, серіалізація та десеріалізація, перевірки відповідності та правильності структури схеми моделі [12]. Використовується веб-фреймворком для обробки даних що пересилаються між користувачем та сервером.
- Sqlalchemy – ORM для взаємодії з СУБД, створення програмних сутностей. Полегшує роботу з БД, адже прибирає необхідність реалізовувати власні SQL-команди, змінювати їх за зміни вимог.
- Alembic – Інструмент для роботи з міграціями даних СУБД. Довзоляє описувати етапи перенесення даних між базами даних старих форматів, реалізовує сам процес перенесення цих даних.

Проект сервісу поділений на наступні модулі:

- models – Сутності що використовуються при роботі з СУБД.
- request_models – Сутності що використовуються при роботі з веб-запитами, для генерації документації.
- middlewares – Методи перевірки правдивості JWT-токена від сервісу авторизації.
- alembic – Скрипти міграцій схеми СУБД.
- routes – Методи обробки веб-запитів.
- permissions – Методи перевірки наявності у користувача прав доступу.
- utils – Додаткові методи та класи що використовуються у окремих компонентах сервісу.
- tests – Методи тестування.

5.5. Сервіс агрегації та аналізу даних

Цей сервіс відповідає за роботу з великою кількістю даних. До сфер його застосування належить (рисунок 5.7):

- Експортування даних:
  - Встановлення фільтрів, сортування.
  - Вибір необхідних полів, їх порядку.
- Імпортування даних:
  - Вибір необхідних полей імпортування.
  - Вирішення конфліктів імпорту.
- Отримання агрегованих даних:
  - Вибір параметрів агрегування даних:
    - Період (30 хвилин, 1 год, 1 день, 1 тиждень).
    - Параметри що необхідно знайти за кожний період.
    - Тип інтерполяції даних (Відсутня/Лінійна).

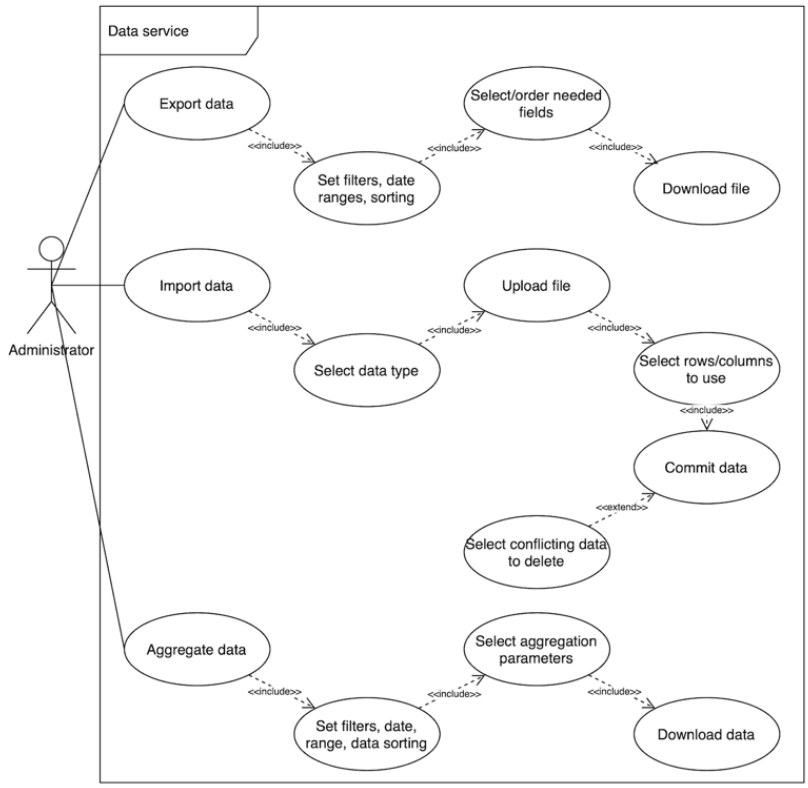


Рисунок 5.7 – Прецеденти сервісу агрегації та аналізу даних

Через вимогливість до сервісу у зв’язку із великими об’ємами даних що проходять через нього, сервер було реалізовано на мові Java із використанням наступного стеку технологій:

- Spring Boot – Веб-фреймворк для розробки застосунків. Містить наступні підмодулі:
    - Core – Ключова частина, на котрій базуються інші модулі.
    - Data – Модулі роботи з БД/інтеграції з ORM:
        - JDBC.
        - Transactions.
    - Web – Модулі веб–застосунків:
        - Web.
        - Servlet.
    - Test – Модулі для написання тестів.
- Reactor project – Бібліотека для роботи з Spring у реактивному стилі.
- Hibernate – ORM для роботи з БД.
- Lombok – Бібліотека для автогенерації методів сутностей.
- Springfox – Бібліотека для автогенерації API для JSON-REST методів.
- OpenCSV – Бібліотека для роботи з файлами типу CSV.
- Apache POI-HSSF – Бібліотека для роботи з файлами типу xlsx.

Проект має наступні модулі:

- controller – Контролерами з методами REST-API.
- exception – Сутності помилок.
- mapper – Класи та методи для роботи з даними схожих об’єктів.
- config – Налаштування веб-серверу.
- model – Сутності системи:
    - files – Файли.
    - entity – Суто програмні сутності.
    - dto – Сутності при взаємодії по мережі.
- util – Методи роботи з рядковими типами.

5.6. WEB застосунок

Веб-застосунок було написано мовою TypeScript + Javascript із використанням технології розробки динамічних веб-застосунків Angular. Також для дизайну використовувалися технології HTML та CSS.

Веб-застосунок реалізовує усі прецеденти закладені серверною частиною, а тому він має дуже великий об’єм. Для роботи з таким об’ємом коду було важливим створення прозорої та легкої для розуміння архітектури.

Проект розділений на наступні категорії:

- models – Сутності що відображаються в інтерфейсі. В свою чергу розбивається на теки за категоріями розділів інтерфейсу.
- core – Сервіси, що відповідають за роботу з сутностями, взаємодію з веб-сервером, також розбиті за категоріями.
- effects, reducers – Методи та класи що відповідають за роботу із станом програми, її збереженням.
- base-core – Суто утилітарні методи для роботи з авторизацією, оновленням токенів, правами користувачів, файлів тощо.
- shared – Опис логіки та вигляду базових компонентів, таких як кнопки, поля вводу, перемикачі, текст, вспливаючі вікна, таблиці, графіки.
- src/app – Містить усі екрани, компоненти що є унікальними для них. Є найбільшим розділом. Узагальнюючи, можна навести наступні приклади:
    - Основна структура та навігація.
    - Сторінка з загальною інформацією про стан.
    - Авторизація, Користувачі, групи.
    - Місця, будівлі, поверхи, кімнати.
    - Контроль температур.
    - Документаційні розділи.
    - Лічильники та їх показники.

5.7. Мобільний застосунок

Мобільний застосунок написано мовою Dart з використанням технології кросплатформенної розробки графічних застосунків Flutter. Основною ціллю мобільного застосунку є взаємодія з наступними елементами системи:

- Документація.
- Тарифи.
- Лічильники, знімки показників лічильників.
- Кімнати, облік обладнання, знімки показників середовища.

Перед створенням мобільного застосунку, було сплановано та розроблено відповідну діаграму сценаріїв користування (UX Flow) (рисунок 5.8). Отримана при аналізі інформація була частково вживана вже за розробки програми.

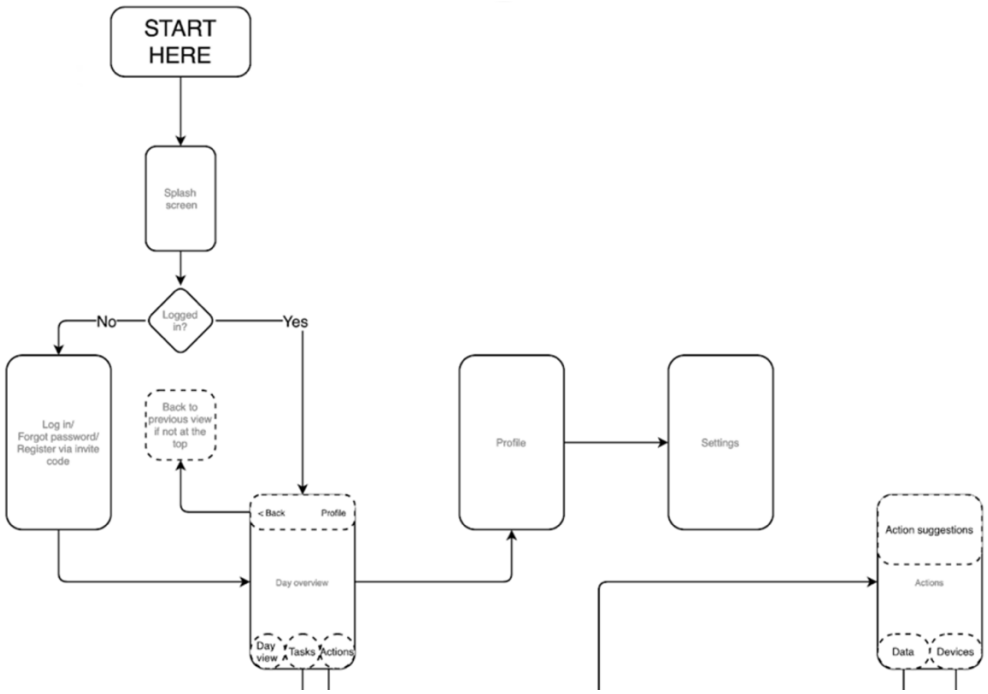


Рисунок 5.8 – Фрагмент діаграми сценаріїв користування

Чітко визначена сфера використання мобільного застосунку дозволяє залишити його сфокусованим та простим для кінцевого користувача, це досягнуто завдяки тому що інтерфейс не містить зайвих елементів що будуть маловживаними у промодельованих сценаріях використання.

Висновки до розділу 5

У цьому розділі було описано структуру застосунку, визначено та зображено архітектуру загальної системи, виділено її окремі компоненти:

1. Визначено елементи Backend-системи, детально розглянуто її складові елементи:
    a. Описано прецеденти, сутності, функціонал сервісу авторизації. Зазначено стек використаних технологій, структуру проекту та розроблених модулів.
    b. Визначено сутності та вимоги сервісу документообігу. Описано спільні аспекти прецедентів системи, визначено технічні особливості цього сервісу.
    c. Розглянуто сервіс метрики, описано його сферу відповідальності, головні сутності, елементи функціоналу. Зазначено використані технології та структуру модулю.
    d. Описано сервіс агрегації даних. Зазначено основну особливість цього сервісу, що полягає у вимогах до оптимізації, та мову і технології розробки що були використані через це. Розглянуто прецеденти і основний функціонал.
2. Дано опис користувацьким застосункам:
    a. Описано функціонал та архітектуру Web-застосунку. Зазначено використані бібліотеки та фреймворки.
    b. Визначено цільову аудиторію, сферу використання мобільного застосунку. Розроблено сценарії використання, на їх основі проведено оптимізацію вимог.

Підводячи підсумок, для усіх компонентів було описано наявні вимоги, функціонал, сутності, технічні інструменти розробки що використовувалися, зазначено структуру модулей застосунку, використані залежності, фреймворки та бібліотеки та ціль їх використання при розробці.

6. ВИПРОБУВАННЯ ТА ДОКУМЕНТУВАННЯ

Перед наданням системи на використання кінцевим користувачам, необхідно провести її ретельну перевірку методом тестування, як і автоматизованим, так і мануальним.

6.1. Тестування веб-сервісів

Для тестування левової частки функціоналу веб-сервісів використовуються автоматизовані тести, що були розроблені разом із основною системою. Такі тести передбачаються для запобіганню наступним проблемам:

- Логічні помилки реалізації.
- Виникнення неочікуваної поведінки.
- Зміни результатів при рефакторингу, модифікації коду.

Перед початком інтеграції користувацьких застосунків із серверною частиною, було проведено мануальне та автоматизоване тестування API кожного з розроблених сервісів. Для цього застосовувався графічний застосунок Swagger, що генерує інтерфейс для взаємодії з застосунком по наданій специфікації OpenAPI (рисунок 6.1). Swagger надає можливість перевіряти наступний функціонал:

- Авторизація.
- Система токенів.
- GET/POST/PUT/DELETE/* запити.
- Помилки.
- Формати запиту, типи даних, файлів, тощо.
- Заголовки запитів.
- Структура відповідей, запитів.
- Перелік всіх сутностей сервера.

Рисунок 6.1 – Інтерактивний блок для тестування REST–запиту

Під час проведення мануального тестування API було винайдено та виправлено наступні класи помилок:

- Відсутні методи запитів:
  - Видалення.
  - Створення.
  - Перегляду.
- Друкарські помилки у назвах.
- Невідповідність назв полів/сутностей до визначеної специфікації.
- Відсутність деяких сутностей.
- Проблеми з заголовками запитів.
- Відсутність підтримки певних типів файлів.
- Програмні помилки що призводять до помилки сервера, втрати даних, виходу системи з ладу.

Оперативне знаходження, пошук, аналіз та виправлення помилок допомогло зберегти час на розробку, що був витрачений розробниками користувацьких застосунків.

6.2. Перевірка користувацьких застосунків

Графічні застосунки є надважливими у будь-якій системі, що націлена на використання кінцевим користувачем. Через це до їх тестування було приділено велику кількість уваги.

Основною методикою при перевірці графічних застосунків виступало мануальне тестування. Кожен із розроблених екранів було переглянуто “з боку користувача”, інтерактивні елементи інтерфейсу було перевірено на відповідність функціоналу, виконано ключові дії що відносяться до кожної з сторінок. Було запрошено тестових користувачів для ознайомлення та відгуку щодо вигляду та функціоналу реалізованих застосунків.

Як наслідок тестування, було знайдено та виправлено такі класи помилок:

- Друкарські помилки.
- Відсутність локалізованої назви, відображення технічних назв у нетехнічних вікнах (рисунок 6.2).
- Дубльовані сторінки.
- “Мертві” посилання.
- Графічні помилки (рисунок 6.3):
    - Неправильні кольори.
    - Неправильно розташовані компоненти.
    - Overflow-помилки.
- Непрацюючі кнопки, сторінки.
- Програмні помилки:
    - Дублювання даних.
    - Відсутність змінних.
    - Неправильне форматування даних.
    - Вічні цикли, що навантажують пристрій.

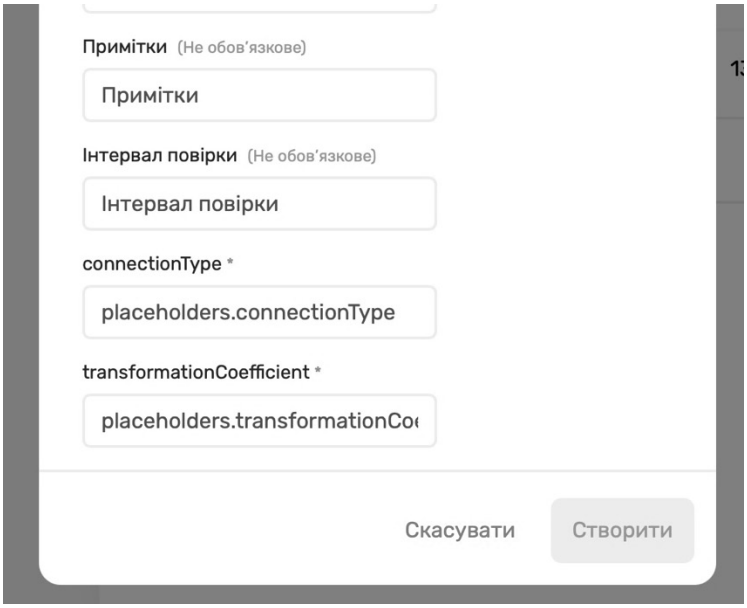


Рисунок 6.2 – Приклад відсутнього перекладу

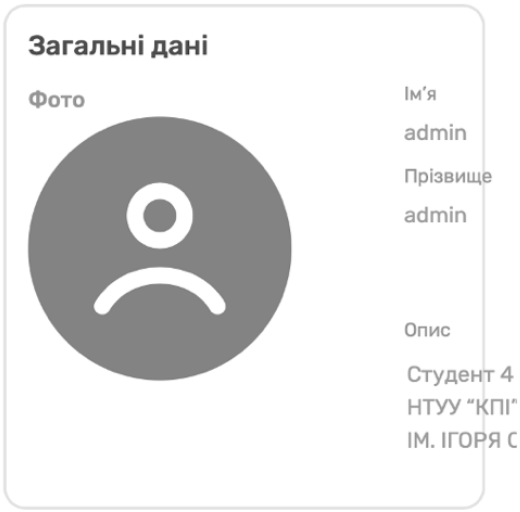


Рисунок 6.3 – Приклад графічної помилки

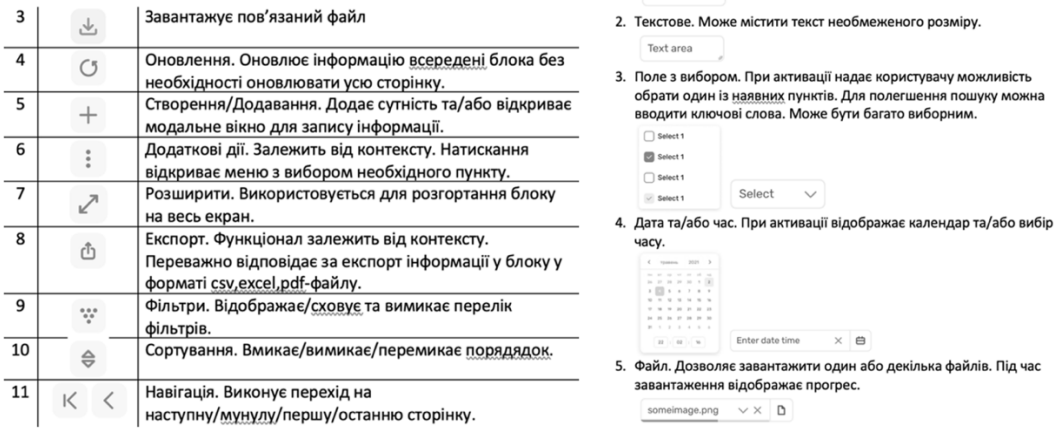
Окрім функціональних та графічних помилок, з аналізу досвіду користувачів та розробників що виконували тестування, було зроблено висновки щодо окремих дизайнерських та графічно-структурних рішень [8]. Як наслідок, було виконано відповідні зміни до специфікації вимог графічного інтерфейсу. Таким чином, наприклад, деякі екрани було значно змінено, спростовано, об'єднано з іншими, а деякі графічні компоненти набули змін для підвищення легкості їх впізнання, взаємодії з ними.

6.3. Розробка документації

Системи що охоплюють велику кількість функціоналу повинні мати відповідно розроблену документацію, адже саме документація повинна надати користувачеві первинну допомогу по використанню системи.

У документації, що була розроблена для системи, було розглянуто наступні аспекти та пункти:

- Графічний застосунок:
    - Загальна структура.
    - Компоненти (рисунок 6.4):
        - Введення інформації (Поля, перемикачі тощо).
        - Компоненти взаємодії (Кнопки).
        - Комплексні:
            - Таблиці.
            - Графіки.
    - Екрани, сторінки:
        - Модальні вікна:
            - Створення, видалення.
            - Підтвердження дії.
        - Сторінки сутностей.
        - Сторінки детального перегляду.
- Система, її специфікація:
    - Система груп, користувачів, запрошень.
    - Сутності системи.
- Встановлення, первинне налаштування:
    - Налаштування серверу.
    - Перенесення файлів.
    - Запуск застосунку.
    - Створення користувачів та груп.



3		Завантажує пов'язаний файл
4		Оновлення. Оновлює інформацію всередені блока без необхідності оновлювати усю сторінку.
5		Створення/Додавання. Додає сутність та/або відкриває модальне вікно для запису інформації.
6		Додаткові дії. Залежить від контексту. Натискання відкриває меню з вибором необхідного пункту.
7		Розширити. Використовується для розгортання блоку на весь екран.
8		Експорт. Функціонал залежить від контексту. Переважно відповідає за експорт інформації у блоку у форматі csv,excel,pdf-файлу.
9		Фільтри. Відображає/сховує та вимикає перелік фільтрів.
10		Сортування. Вмикає/вимикає/перемикає порядядок.
11		Навігація. Виконує перехід на наступну/мунулу/першу/останню сторінку.

Рисунок 6.4 – Уривки зі сторінок документації

У документації робився окремий акцент на використання відповідного графічного матеріалу, що відображає реальні приклади, аби запобігти виникненню плутанини. Елементи складних компонентів нумеруються, по кожному із номерів надається окреме пояснення.

Під час розробки документації було отримано відгук від користувачів, виконано відповідні зміни, внесені більш чіткі описи щодо бізнес процесів, екранів взаємодії з сутностями.

Висновки до розділу 6

У цьому розділі було розглянуто проблематику тестування програмного забезпечення, описано типи тестування та процеси їх проведення. Запропоновано методики автоматизованого та ручного тестування. Після проведення тестування було визначено проблемні моменти, запропоновані відповідні рішення. Як наслідок, система стала більш стійкою до помилок, були внесені значні покращення до алгоритмів, структури застосунку.

По завершенню основного етапу розробки застосунку, було складено детальну документацію до системи, описані компоненти інтерфейсу, екрани користувачів. Виконано детальний опис процесу первинного налаштування та встановлення. Враховуючи відгуки залучених до розробки адміністраторів служби енергоменеджменту, документація була дороблена та покращена.

ВИСНОВКИ

Під час виконання дипломної роботи були виконані наступні етапи розробки програмного продукту:

1. Проведено аналіз вимог до системи, розглянуто існуючі рішення, виконано висновки на базі отриманої інформації..
2. На базі отриманих вхідних даних, було розроблено загальну специфікацію системи.
3. Спроектована архітектура та сутності серверної частини системи, розроблена специфікація алгоритмів.
4. Розроблено дизайн-систему та макети графічного інтерфейсу користувача.
5. Виконана декомпозиція поставленої задачі. Завдання були розподілені між членами команди розробки.
6. Розроблено методологію автоматичного та мануального тестування, виконано тестування системи з точки зору користувача.
7. Складена деталізована документація користувача.

Розроблена система працює справно, поставлені на першу ітерацію розробки задачі були вправно виконані. До продукту надано доступ зацікавленим особам, отримано відгуки щодо бажаних вдосконалень та покращень, виправлені деякі помилки. Враховуючи зібрані побажання, застосунок може бути розширеним та покращеним у наступних ітераціях розробки.

Під час виконання поставленої задачі, були отримані навички аналізу системи, її проектування, розробки дизайну та графічного інтерфейсу, командної взаємодії та комунікації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Основи проектування та використання баз даних: Навч. посібник – 2-ге [посібник] вид., виправл. і допов./ В. І. Гайдаржи, О. А. Дацюк – К.:ІВЦ "Видавництво "Політехніка", ТОВ "Фірма "Періодика", 2004. — 172 с.
2. Jooby [електронний ресурс] - режим доступу до ресурсу: https://jooby.eu/ .
3. VSENERGY [електронний ресурс] - режим доступу до ресурсу: https://vsenergy.com.ua/wp-content/uploads/2018/09/luzodascoe2.pdf .
4. IKNET: АСКОЕ [електронний ресурс] – режим доступу до ресурсу: https://iknet.com.ua/uk/ .
5. JWT [електронний ресурс] – режим доступу до ресурсу: https://jwt.io/introduction .
7. Кнут Д.Е. Майстерність програмування [Книга] / під ред. С. Г. Тригуб (гл. 1), Ю. Г. Гордієнко (гл. 2) та І. В. Красикова (разд. 2.5 и 2.6). — 3. — // Москва: Вільямс, 2002. ISBN 978-5- 8459-0082-1.
8. Jennifer N.R. "Learning Web Design" [посібник] // N.R Jennifer // Sebastol C.A.: O'Reilly Media // 2012. – 120с .
9. Figma [електронний ресурс] – режим доступу до ресурсу: https://www.figma.com/ .
10. JetBrains [електронний ресурс] – режим доступу до ресурсу: https://www.jetbrains.com/ .
11. Django [електронний ресурс] – режим доступу до ресурсу: https://www.djangoproject.com .
12. FastApi [електронний ресурс] – режим доступу до ресурсу: https://fastapi.tiangolo.com .
13. Angular [електронний ресурс] – режим доступу до ресурсу: https://angular.io .

ДОДАТОК А

Програмний модуль
системи моніторингу споживання електричної енергії

Специфікація

УКР.НТУУ«КПІ»_ТЕФ_АПЕПС_ТІ71102_21А 1

Аркушів 2

Київ – 2021

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ71102_21Б 2	Записка Коротич М.С. ТІ-71.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ71102_21Б 2-1	contacts_service.ts	Компонент керування контактними даними
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ71102_20Б 2-2	manage_users_service.ts	Компонент керування користувачами
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ71102_20Б 2-3	groups_service.ts	Компонент керування групами
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ71102_20Б 2-4	rooms _service.ts	Компонент керування кімнатами
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТІ71102_20Б 2-5	buildings_service.ts	Компонент керування будинками

ДОДАТОК Б

Програмний модуль
системи моніторингу споживання електричної енергії

Лістинг програми

УКР.НТУУ«КПІ»_ТЕФ_АПЕПС_ТІ71102_21Б 2

Аркушів 6

Київ – 2021

contacts_service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';

import { CamelCaseHelperService } from '@base-core';

import { ContractModel, PaginationModel } from '@models';

import { ApiUrls } from '../api-urls';

@Injectable()
export class ContractsService {
 constructor(
  private httpClient: HttpClient,
  private camelCaseHelper: CamelCaseHelperService
 ) { }

 public getContracts(): Observable<PaginationModel<ContractModel>> {
  return this.httpClient.get<PaginationModel<ContractModel>>(ApiUrls.getContractsUrl())
   .pipe(
    map(contracts => ({
     ...contracts,
     items: contracts.items.map(c => new ContractModel(c))
    }))
   );
 }

 public createContract(doc: ContractModel): Observable<ContractModel> {
  const data: FormData = new FormData();

  doc.startDate = new Date(doc.startDate).toISOString();
  doc.expirationDate = new Date(doc.expirationDate).toISOString();

  Object.keys(doc).forEach(k => data.set(this.camelCaseHelper.toSnakeCase(k), doc[k as keyof ContractModel] as File));

  return this.httpClient.post<ContractModel>(ApiUrls.getContractsUrl(), data);
 }

 public editContract(doc: ContractModel): Observable<ContractModel> {
  const data: FormData = new FormData();

  console.log(doc);

  doc.startDate = new Date(doc.startDate).toISOString();
  doc.expirationDate = new Date(doc.expirationDate).toISOString();

  Object.keys(doc).forEach(k => data.set(this.camelCaseHelper.toSnakeCase(k), doc[k as keyof ContractModel] as File));

  return this.httpClient.patch<ContractModel>(ApiUrls.getContractsUrl(doc.id.toString() + '/'), data);
 }

 public removeContract(docId: string): Observable<null> {
  return this.httpClient.delete<null>(ApiUrls.getContractsUrl(docId + '/'));
 }
}
```

manage_users_service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import { combineLatest, Observable, of } from 'rxjs';

import { AddUserModel, ContactInfoModel, CrudInterface, InvitedUserModel, PaginationModel, QueryModel, UserProfileModel } from '@models';

import { ApiUrls } from '../api-urls';
import { concatMap, map } from 'rxjs/operators';

@Injectable()
export class ManageUsersService implements CrudInterface {
 constructor(
```

```
  private httpClient: HttpClient
) {
}

public create(args: any): Observable<any> {
  return of(null);
}

public delete(args: any): Observable<any> {
  return of(null);
}

public read(query?: QueryModel<UserProfileModel>): Observable<PaginationModel<UserProfileModel>> {
  return this.httpClient.get<PaginationModel<UserProfileModel>>(ApiUrls.getUsersUrl(), {
    params: query?.httpParams
  });
}

public update(args: any): Observable<any> {
  return of(null);
}

public getUsers(): Observable<PaginationModel<UserProfileModel>> {
  return this.httpClient.get<PaginationModel<UserProfileModel>>(ApiUrls.getUsersUrl());
}

public getUser(id: number): Observable<UserProfileModel> {
  return this.httpClient.get<UserProfileModel>(ApiUrls.getUserByIdUrl(id));
}

public getInvitedUsers(): Observable<PaginationModel<InvitedUserModel>> {
  return this.httpClient.get<PaginationModel<InvitedUserModel>>(ApiUrls.getUserInvitesUrl())
    .pipe(
      map((users: PaginationModel<InvitedUserModel>) => ({
        ...users,
        items: users.items.map(user => new InvitedUserModel(user))
      }))
    );
}

public createUser(user: AddUserModel): Observable<InvitedUserModel> {
  const contactInfos: ContactInfoModel[] = user?.contactInfos.filter(info => Object.values(info).every(item => !!item));

  if (user.photo) {
    const formData: FormData = new FormData();
    formData.append('photo', user.photo);

    delete user.photo;

    return this.httpClient.post<InvitedUserModel>(ApiUrls.getUsersAddUrl(), {
      ...user,
      contactInfos
    })
      .pipe(
        concatMap(({invitee}) => this.httpClient.patch<InvitedUserModel>(ApiUrls.getUserByIdUrl(invitee.id!), formData))
      );
  }

  return this.httpClient.post<InvitedUserModel>(ApiUrls.getUsersAddUrl(), {
    ...user,
    contactInfos
  });
}

public editUser(user: AddUserModel): Observable<InvitedUserModel> {
  const contactInfos: ContactInfoModel[] = user?.contactInfos.map(info => !!info?.id ? info : {
    name: info.name,
    type: info.type,
    value: info.value,
    notes: info.notes
  });

  if (user.photo) {
    const formData: FormData = new FormData();
    formData.append('photo', user?.photo as File);

    delete user.photo;
```

```
    return this.httpClient.patch<InvitedUserModel>(ApiUrls.getUserByIdUrl(user.id!), formData)
      .pipe(
       concatMap(_ => this.httpClient.patch<InvitedUserModel>(ApiUrls.getUserByIdUrl(user.id!), {
         ...user,
         contactInfos
       }))
      );
   }

   return this.httpClient.patch<InvitedUserModel>(ApiUrls.getUserByIdUrl(user.id!), {
    ...user,
    contactInfos
   });
  }

  public deleteUser(userId: number): Observable<null> {
   return this.httpClient.delete<null>(ApiUrls.getUserByIdUrl(userId));
  }
}
```

groups_service.ts

```
import { Injectable } from '@angular/core';
import { AddUserToGroupModel, CrudInterface, GroupModel, PaginationModel, QueryModel, UserRoleInGroupEnum } from '@models';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { ApiUrls } from '../api-urls';
import { map, tap } from 'rxjs/operators';

@Injectable()
export class GroupsService implements CrudInterface {
  constructor(
   private httpClient: HttpClient
  ) {
  }

  public read(query?: QueryModel<GroupModel>): Observable<PaginationModel<GroupModel>> {
   return this.httpClient.get<PaginationModel<GroupModel>>(ApiUrls.getUsersGroupsUrl(), {
    params: query?.httpParams
   })
    .pipe(
     map(groups => ({
      ...groups,
      items: groups.items.map((g: GroupModel) => new GroupModel(g))
     }))
    );
  }

  public create(group: GroupModel): Observable<GroupModel> {
   return this.httpClient.post<GroupModel>(ApiUrls.getUsersGroupsUrl(), group);
  }

  public delete(group: GroupModel): Observable<null> {
   return this.httpClient.delete<null>(ApiUrls.getUsersGroupsUrl(group.id!.toString() + '/'));
  }

  public update(group: GroupModel): Observable<GroupModel> {
   return this.httpClient.patch<GroupModel>(ApiUrls.getUsersGroupsUrl(group.id!.toString() + '/'), group);
  }

  public removeUserFromGroup(groupId: number, userId: number): Observable<null> {
   return this.httpClient.delete<null>(ApiUrls.getUsersGroupsRemoveUserUrl(groupId, userId));
  }

  public addUserToGroup(user: AddUserToGroupModel, groupId: number): Observable<null> {
   if (user.userRole === UserRoleInGroupEnum.admin) {
    return this.httpClient.post<null>(ApiUrls.getUsersGroupsAddAdminUrl(groupId), user);
   }

   return this.httpClient.post<null>(ApiUrls.getUsersGroupsAddUserUrl(groupId), user);
  }
}
```

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
```

rooms_service.ts

```
import { Observable, of } from 'rxjs';

import { CrudInterface, PaginationModel, QueryModel, RoomModel } from '@models';

import { ApiUrls } from '../api-urls';
import { take } from 'rxjs/operators';

@Injectable()
export class RoomsService implements CrudInterface {
  constructor(
    private httpClient: HttpClient
  ) { }

  public read(query?: QueryModel<RoomModel>): Observable<PaginationModel<RoomModel>> {
    return this.httpClient.get<PaginationModel<RoomModel>>(ApiUrls.getRoomsUrl(), {
      params: query?.httpParams
    }).pipe(take(1));
  }

  public delete(args: any): Observable<any> {
    return of(null);
  }

  public create(args: any): Observable<any> {
    return of(null);
  }

  public update(room: RoomModel): Observable<RoomModel> {
    return this.editRoom(room);
  }

  public getRooms(): Observable<PaginationModel<RoomModel>> {
    return this.httpClient.get<PaginationModel<RoomModel>>(ApiUrls.getRoomsUrl());
  }

  public createRoom(room: RoomModel): Observable<RoomModel> {
    return this.httpClient.post<RoomModel>(ApiUrls.getRoomsUrl(), room);
  }

  public editRoom(room: RoomModel): Observable<RoomModel> {
    return this.httpClient.patch<RoomModel>(ApiUrls.getRoomsUrl(room.id!.toString()), room);
  }

  public deleteRoom(roomId: string): Observable<null> {
    return this.httpClient.delete<null>(ApiUrls.getRoomsUrl(roomId + '/'));
  }
}
```

buildings_service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import { Observable, of } from 'rxjs';

import { AddBuildingModel, BuildingModel, BuildingTypeModel, CrudInterface, HeadcountModel, PaginationModel, QueryModel } from '@models';

import { ApiUrls } from '../api-urls';
import { map } from 'rxjs/operators';

@Injectable()
export class BuildingService implements CrudInterface {
  constructor(
    private httpClient: HttpClient
```

```
  ) { }

  public create(building: BuildingModel): Observable<any> {
    return this.createBuilding(building);
  }

  public delete(building: BuildingModel): Observable<any> {
    return this.removeBuilding(building.id);
  }

  public read(query?: QueryModel<BuildingModel>): Observable<PaginationModel<BuildingModel>> {
    return this.getBuildings(query);
  }

  public update(building: BuildingModel): Observable<any> {
    return this.editBuilding(building);
  }

  public getBuildingTypes(): Observable<PaginationModel<BuildingTypeModel>> {
    return this.httpClient.get<PaginationModel<BuildingTypeModel>>(ApiUrls.getBuildingTypesUrl());
  }

  public getBuildingTypesCount(): Observable<PaginationModel<BuildingTypeModel>> {
    return this.httpClient.get<PaginationModel<BuildingTypeModel>>(ApiUrls.getBuildingTypesCountUrl());
  }

  public createBuildingType(name: string): Observable<BuildingTypeModel> {
    return this.httpClient.post<BuildingTypeModel>(ApiUrls.getBuildingTypesUrl(), {name});
  }

  public editBuildingType(building: BuildingTypeModel): Observable<BuildingTypeModel> {
    return this.httpClient.patch<BuildingTypeModel>(ApiUrls.getBuildingTypesUrl(building.id.toString()), {
      name: building.name
    });
  }

  public removeBuildingType(buildingId: number): Observable<null> {
    return this.httpClient.delete<null>(ApiUrls.getBuildingTypesUrl(buildingId.toString() + '/'));
  }

  public getBuildings(query?: QueryModel<BuildingModel>): Observable<PaginationModel<BuildingModel>> {
    return this.httpClient.get<PaginationModel<BuildingModel>>(ApiUrls.getBuildingUrl(), {
      params: query?.httpParams
    })
      .pipe(
        map(b => ({
          ...b,
          items: b.items.map(item => new BuildingModel(item))
        }))
      );
  }

  public createBuilding(building: BuildingModel): Observable<BuildingModel[]> {
    return this.httpClient.post<BuildingModel[]>(ApiUrls.getBuildingUrl(), new AddBuildingModel(building));
  }

  public editBuilding(building: BuildingModel): Observable<BuildingModel[]> {
    return this.httpClient.patch<BuildingModel[]>(ApiUrls.getBuildingUrl(building.id.toString()), building);
  }

  public removeBuilding(buildingIid: number): Observable<null> {
    return this.httpClient.delete<null>(ApiUrls.getBuildingUrl(buildingIid.toString() + '/'));
  }

  public getHeadcount(): Observable<HeadcountModel> {
    return this.httpClient.get<HeadcountModel>(ApiUrls.getHeadCountUrl());
  }
}
```