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

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

"На правах рукопису"
УДК 004.4

«До захисту допущено»
В.о. зав.кафедри
_____ Олександр КОВАЛЬ
“___” _____202_ р.

Магістерська дисертація

За освітньою програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»
Спеціальності 121 Інженерія програмного забезпечення
на тему: Архітектура та патерни мікрофронтендів, принципи їх реалізації, слабкі та сильні сторони

Виконав: студент 2 курсу, групи ТВ-12мп
Веретьонкін Олексій Сергійович _____ _____
(прізвище. ім’я, по батькові) (підпис)

Науковий керівник
к.т.н., доцент Гагарін Олександр Олександрович _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2022

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти другий, магістерський
За освітньою програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»

Спеціальності 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
в.о. зав. кафедри
_____ Олександр КОВАЛЬ
(підпис)

«____» _____ 202_р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТУ

Веретьонкіну Олексію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації:
Архітектура та патерни мікрофронтендів, принципи їх реалізації, слабкі та сильні сторони
Науковий керівник к.т.н, доцент Гагарін Олександр Олександрович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від "07"листопада 2022 року №4067-с
2. Строк подання студентом дисертації 05 грудня 2022 року.
3. Вихідні дані до роботи: Мова програмування TypeScript, клієнтська частина додатку «Кабінет енергоменеджера».
4. Перелік питань, які потрібно розробити: Розробити архітектуру мікрофронтендів. Дослідити слабкі та сильні сторони цього підходу. Розробити систему згідно встановленого дизайну. Зробити бібліотеку компонентів. Поділити застосунок на окремі мікрофронтенди що будуть збиратися у один веб-додаток. Розробити кожен мікрофронтенд таким чином щоб його можна було легко розширити, а за необхідності поділити на інші мікрофрнотенти. Реалізувати операції створення, оновлення, читання та видалення сутностей. Налаштувати механізи розгортання системи.
5. Орієнтований перелік ілюстративного матеріалу: Титульна сторінка, Актуальність, Огляд подібних рішень, Діаграма керування станом застосунку

Технології що використовуються, Структура файлів, Інструменти розробки, Вигляд розроленого веб-додатка, Висновки.
6. Дата видачі завдання «01» жовтня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	строки виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	01.10.21	виконано
2	Дослідження предметної області	01.10.21-01.11.21	виконано
3	Постановка вимог до проєктування системи	01.11.21-15.11.21	виконано
4	Дослідження існуючих рішень	15.11.21-01.12.21	виконано
5	Розробка програмного продукту	01.12.2022-07.10.2022	виконано
6	Тестування	07.10.2022-11.11.2022	виконано
7	Захист програмного продукту	17.10.2022-21.10.2022	виконано
8	Підготовка магістерської дисертації	22.10.2022-20.11.2022	виконано
9	Передзахист	21.11.2022-25.11.2022	виконано
10	Захист	19.12.2022-23.12.2022	виконано

Студент _____ Веретьонкін О.С.
(підпис) (прізвище та ініціали)

Науковий керівник _____ Гагарін О.О.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Актуальність теми. Розроблена система дозволяє виконувати моніторинг енергоресурсів. Перехід на джерела альтернативної енергії(сонячні панелі, вітряки, тощо) потребують постійного огляду за енергією що використовуються. Забезпечення моніторингу над ресурсами може показати місця де необхідна термінова заміна обладнання, наприклад: старі лампочки потребують дуже багато електроенергії, або протікання старого крану спричиняє додаткове споживання води.

Мета і задачі дослідження. Метою роботи є побудова клієнтського застосунку що можна буде легко розширювати та додавати туди новий функціонал без необхідності глобального оновлення всього застосунку.

Об'єкт дослідження. Технології що забезпечують розробку клієнтських додатків, процес розробки фронтендів.

Предмет дослідження. Система мікрофронтендів для створення веб-додатку управління системою енергоменеджменту.

Методи дослідження. Побудова архітектури мікрофронтендів.

Практичне значення одержаних результатів. Розроблена система спрощує роботу для енергоменеджерів слугуючи їм інструментом для ведення документобігу та огляду за енергоресурсами. Систему можна впровадити для звичайних будинків щоб можна було вести облік спожитих ресурсів.

Обсяг дипломної роботи. Робота складається із вступу, восьми розділів, висновку, списку використаних джерел із 10 найменувань, 2 додатків. У дисертації наявні 33 рисунка та 9 таблиць. Обсяг роботи ― 87 сторінок.

Ключові слова: фронтенд, мікрофронтент, деплой , стейт менеджмент, модальні вікна, таблиці, client side rendering, server side rendering.

ABSTRACT

Actuality of theme. The developed system allows monitoring of energy resources. The transition to alternative energy sources (solar panels, windmills, etc.) requires a constant review of the energy used. Providing monitoring over resources can show places where urgent equipment replacement is needed, for example: old light bulbs require too much electricity, or a leaking old faucet causes additional water consumption.

The purpose and objectives of the study. The goal of the work is to build a client application that can be easily expanded and add new functionality without the need for a global update of the entire application.

Object of study. Technologies that ensure the development of client applications, the process of developing frontends.

Subject of study. A system of microfrontends for creating a web application for managing the energy management system.

Research methods. Building the architecture of microfrontends.

Practical significance of the obtained results. The developed system simplifies the work of energy managers, serving them as a tool for document management and review of energy resources. The system can be implemented for ordinary houses to keep track of consumed resources.

The scope of the thesis. The work consists of an introduction, eight chapters, a conclusion, a list of used sources from 10 titles, 2 appendices. There are 33 figures and 9 tables in the dissertation. The volume of the work is 87 pages.

Keywords: frontend, microfrontend, deplay, state management, modal windows, tables, client side rendering, server side rendering

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Http запит – запит, надісланий на віддалений сервер для введення або отримання інформації.

Фронтенд – загальнодоступна назва інтерфейсу користувача (наприклад, веб-програми, мобільної програми тощо)

Бекенд – це загальна назва сервера, який забезпечує взаємодію з базами даних, обробляє запити користувачів і виконує бізнес-логіку.

PWA додаток (англ. Progressive Web Applications) – тип веб-програми встановлення на комп'ютері має більше можливостей, ніж звичайні веб-додатки.

API – це набір методів, які серверна частина може використовувати для взаємодії з системою.

ПЗ – програмне забезпечення

Розгортання(Деплой) – процес завантаження програми на серверТокен – ключ, який видається користувачеві після успішної автентифікації

Кінцева точка(англ. Endpoint) – Це кінець каналу зв'язку. Для API кінцева точка – це URL-адреса для доступу до певного ресурсу

SPA(англ. Single Page Application) – це веб-програма, яка використовує один документ HTML для всіх сторінок.

Мікрофронтенд(англ. micro frontend). – це архітектура, яка розглядає веб-програму як набір програм, розроблених окремими командами. У роботі цей термін відповідає за частину застосунку, а саме окремий застосунок або модуль

ВСТУП

Питання оптимального використання ресурсів сьогодні є дуже актуальним. Встановлені більш ефективні лічильники, лампи та батареї. Будинки додатково утеплюють для захисту від опалення. Все це робиться для того, щоб зменшити витрати електроенергії. Кількість великих компаній стрімко зростає, і, залежно від їх типу, збільшується кількість енергоресурсів, які вони використовують.

Оптимізація споживання енергії та збір інформації про використання енергії є дуже складним завданням, уявіть, скільки часу та людей це зайняло б, якби не було центральної системи для введення та відображення інформації. Тому для того, щоб покращити збір інформації, нам потрібна система, яка дозволить фіксувати показання всіх лічильників, датчиків тощо. Такий підхід дозволяє відразу побачити загальну картину, дозволяючи краще контролювати використання ресурсів. Система також повинна мати можливість створювати будівлі з поверхами та кімнатами, щоб точніше контролювати споживання.

Сучасні комп’ютери завжди мають браузер, тому системою керує браузер, таким чином користувачі можуть керувати системою з будь-якого пристрою. Також в системі передбачена установка програм PWA.

Розширення та подальша розробка системи відіграють важливу роль у життєвому циклі кожного проекту, тому було прийнято рішення побудувати систему з використанням мікрофронтендів, що дозволить покращити розробку системи та, в подальшому, дасть змогу поділити розробку використовуючі окремі команди для кожної частини застосунку.

1 ВСТАНОВЛЕННЯ ВИМОГ ДО СИСТЕМИ

1.1 Постанова задачі

Головним завданням програмного забезпечення є моніторинг енерговитрат. Потенційні користувачі системи – енергоменеджери. Система повинна спростити облік лічильників, корпусів наявних в університеті (університет наведено як приклад). Система має мати графічний інтерфейс користувача, що представляє собою веб-додаток, мобільний додаток та сервер що виконує операції акумуляції даних та представлення даних користувачеві.

Розглянемо основні недоліки у відсутності уніфікованої системи, для того щоб можна було встановити завдання які система має виконувати:

- Складне введення та обробка інформації;
- Використання великої кількості різних програм потребує великих витрат на навчання, потребує налаштування для кожної програми та вимагає додаткових дій для передачі інформації між програмами та/або на комп’ютери у вигляді документів;
- При збереженні або обміні інформації шляхом заповнення/передання файлів можлива втрата або випадкова зміна інформації;
- Якщо зберігати файли з даними на комп’ютері можливе їх дублювання, або ж повне видалення;
- Якщо кінцевий користувач не налаштує правильну структуру або окремі файли відсутні, виникне проблема зі створенням зв’язків між файлами;
- Відсутність системи контролю доступу. Користувачі можуть мати доступ лише до всього файлу або не мати можливості вводити дані взагалі.

Отже завданням системи є вирішення вищезазначених проблем та полегшення роботи для енергоменеджерів, впровадження нових способів роботи.

Через те що більшість корпусів університету не обладнано лічильниками що дозволяють збирати інформацію з них через інтернет система повинна мати можливість ручного вводу інформації. Оскільки система передбачає внесення даних, що містять точне розташування корпусів, будинків та інших споруд за допомогою координат, необхідно надати можливість обирати місце розташування на інтерактивній карті.

Найкращим рішенням для такої карти є Google Maps. Тобто у додаток необхідно інтегрувати карти від компанії Google. Потрібно передбачити можливість створення локації та вибору її на карті. Крім того користувачеві необхідно надати можливість створення корпусу або будівлі у довільній формі, тобто можливість створити новий корпус і прикріпити його до певної локації.

1.2 Структура програмного забезпечення

Оскільки програмним продуктом будуть користуватися багато людей(енергоменеджерів) необхідно створити профіль користувача в якому зареєстрований користувач зможе редагувати свої контактні дані, ім’я та прізвище, а також встановлювати власний аватар(фотографію). Процес реєстрації користувача необхідно реалізувати через анонімні посилання для реєстрації, переходячи на які новий користувач зможе заповнити форму реєстрації та увійти в систему. Такі посилання будуть видаватися адміністратором конкретній людині, а у разі переходу за невірним посиланням, або таким яке є недійсним, користувач отримає помилку про хибність посилання. Основна частина веб-застосунку має складатися з трьох основних сторінок, відповідно для системи це три основні напрями, а саме:

- Профіль користувача
- Головна панель
- Панель керування системою

Профіль користувача має містити інформацію про поточного користувача, який увійшов у систему. За допомогою цієї сторінки можна змінювати такі дані користувача як ім’я та прізвище, контактні дані та встановлювати фотографію. Зауважимо що профіль користувача не є пріоритетною задачею, тому його функціонал буде певною мірою обмеженим, проте того функціоналу що буде реалізовано має бути цілком достатньо для роботи з користувачами.

Головна панель має містити загальну інформацію по будівлях, відображати повідомлення системи. Інформація по будівлях на цій сторінці несе переважно оглядовий характер, тобто їх загальну кількість, тощо. Також на цій сторінці необхідно розмістити керування документацією. Документи мають мати свій тип, назву та інші атрибути. В свою чергу ці документи завантажуються у систему як файли щоб мати доступ до них в будь-коли.

Панель керування системою становить найважливішу частину застосунку. За допомогою цієї сторінки відбувається процес внесення та редагування даних. Уся інформацію, наявна у цьому розділі має відображатися за допомогою таблиць відповідно до дизайну. Таблиця - як основний інструмент для роботи з даними має надавати користувачу можливість переглядати дані, відкривати модальні вікна для редагування сутносні що була вибрана, та дозволяти видаляти існуючі записи, проте перед видаленням користувача треба попередити про невідворотність видалення інформації.

1.3 Технічна складова системи

З технічної точки зору основне завдання при розробці веб-додатку - це необхідність забезпечити можливість розширювати функціонал вже існуючої програми. Таким чином розширення не повинно ламати вже написану логіку застосунку. До цього необхідно розробити систему таким чином щоб додавання нової логіки або функціоналу в існуючі розділи застосунку не спричиняли його повний перепис чи рефакторинг.

Варто також дотримуватись різних конвенцій при написанні коду, шаблонів проектування та чистої архітектури. Оскільки застосунок в подальшому розглядатиметься як стартап проект вибір технологій на яких він написаний відіграє, якщо не ключову, то дуже важливу роль. Якщо припустити що стартап проект буде успішним може виникнути питання про розширення команди розробників, а це в свою чергу веде за собою їх відбір. В такому випадку треба використовувати новітні та перспективні технології, щоб легше знайти людину що буде працювати над проектом. Варто зазначити що знайти людей на проект зі старими технологіями та легасі кодом дуже складне завдання, тому треба застосовувати переважно ті технології, що користуються популярністю на ринку.

Важливим завданням є можливість працювати декільком людям одночасно. Мікрофронтендна архітектура, яка досліджується у цій роботі, забезпечать виконання цієї вимоги. Таким чином необхідно лише обрати фреймворк як основу реалізації даної архітектури.

Висновки до розділу 1

Отже, головним завданнями є:

1. Аналіз подібних систем з виявленням їх переваг та недоліків.
2. Розробка веб-застосунку згідно існуючого дизайну та структури.
3. Реалізація модульності(поділ на мікрофронтенди) застосунку.
4. Можливість додавання тестів у систему.
5. Розробка механізму розгортання.

2 АНАЛІЗ ПОДІБНИХ ПРОГРАМНИХ РІШЕНЬ

Перш за все при розробці такої системи необхідно ознайомитись з подібними системами управління, щоб мати можливість використовувати систему керування живленням найбільш зручним способом. Зауважте, що жодна окрема система не відповідає всім вимогам, але схожі віддалені системи доступні для дослідження. Загалом усі альтернативні системи підтримують IoT (Internet of things) і в основному використовуються для управління приватними будинками. Крім того, неможливо контролювати всю будівлю та додавати в систему будівлі. Крім того, альтернативні системи не передбачають можливості введення інформації в систему, гнучкого доступу вручну або групою. Така різниця між існуючими рішеннями та системами, що розробляються, пояснюється специфікою програмного продукту (програмне забезпечення в основному призначене для вдосконалення енергоменеджменту в університетах) та вимогами до постачання.

2.1. Jooby

Одним із аналогічних продуктів розробленої системи є Joobi. Joobi встановлює розумне освітлення та автоматизує зчитування лічильників електроенергії, газу, тепла та води. Компанія розробила власну службу збору та обробки даних про енергоспоживання під назвою Jobi RDC Dashboard і Jobi CMS Lighting Management System (рис. 2.1, 2.2).

Програмне забезпечення Joby RDC Dashboard дуже схоже на систему, яка розробляється, тому розглянемо його. Система має такі особливості:

1. Швидке завантаження звітів про споживання ресурсів і стан пристрою у форматі CSV для роботи з іншими програмами, такими як Excel.
2. Окрема сторінка для кожної точки вимірювання з історією показань, лічильників, манометрів і даних споживачів.
3. Інформація про рівень заряду пристрою та стан мережі для зниження експлуатаційних витрат.
4. Гнучке налаштування місця розташування та типу обладнання.
5. Журнал подій: повідомлення про помилки пристрою та мережі.

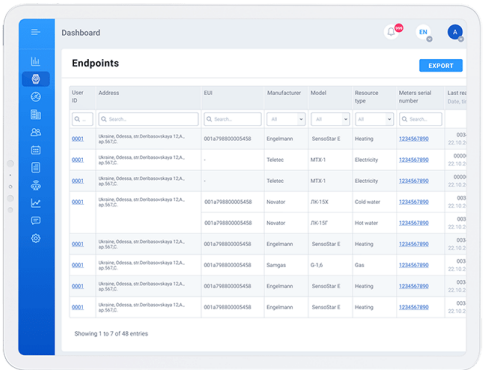


Рисунок 2.1 – Інтерфейс застосунку Jooby RDC Dashboard

Компанія впровадила автоматизацію для зняття показань лічильників електроенергії та води в багатоквартирному комплексі, який раніше знаходився

в Одесі, Україна. Завданням цього проекту було скорочення витрат керуючих компаній на розрахунок енергоресурсів, автоматизація систем збору даних та забезпечення своєчасної дистанційної передачі. Завдяки впровадженню системи зменшено витрати на облік ресурсів в керуючій компанії, з'явилася можливість швидко отримувати точні виміри облікової техніки, оперативно балансувати та формувати звіти. Мешканці та власники офісів отримали корисний інструмент для отримання та надсилання показань.

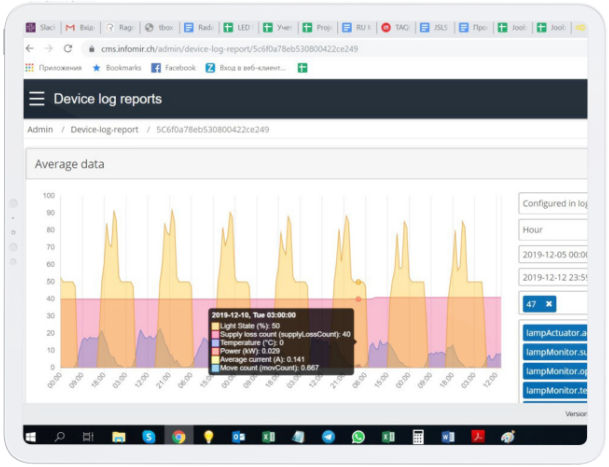


Рисунок 2.2 – Інтерфейс застосунку Jooby CMS

Як бачимо, використання системи позитивно вплинуло на енергоефективність житлових приміщень. Це означає, що такі системи можуть допомогти контролювати ресурси та зменшити витрати на енергію.

Одним із недоліків цієї системи є те, що її не можна використовувати в старих будівлях, які не мають автоматизованих систем зчитування. Оскільки система не має можливості ручного введення інформації, її неможливо використовувати для моніторингу енергоспоживання таких будівель, а встановлення відповідного обладнання призведе до додаткових економічних втрат. Однак у деяких випадках це неможливо.

2.2. Amazon alexa Energy Dashboard

У вересні 2021 року Amazon анонсувала нову інформаційну панель Alexa Energy, до якої можна отримати доступ через додаток Alexa. Це дозволяє клієнтам відстежувати та керувати енергією, споживаною їхніми розумними домашніми пристроями з підтримкою Alexa, в одному місці (рис. 2.3).

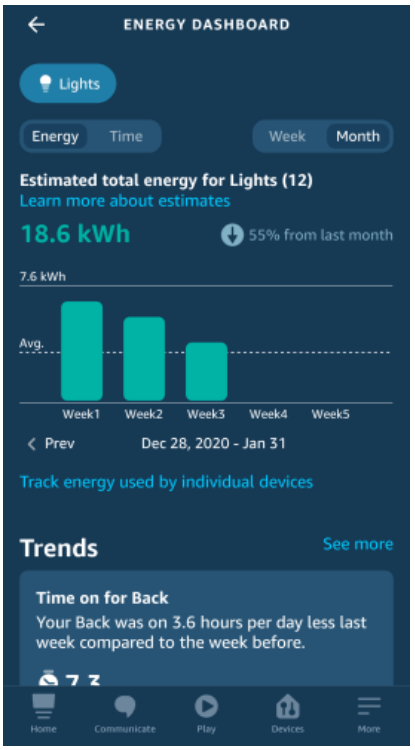


Рисунок 2.3 – Вигляд застосунку Alexa Energy Dashboard

Система дозволяє інтегрувати сумісні світильники, вимикачі, розетки, телевізори, водонагрівачі та термостати в інформаційну панель New Energy, збільшуючи кількість підтримуваних пристроїв. Виробники пристроїв можуть використовувати API продуктів, щоб активно інформувати Alexa про енергоспоживання свого пристрою або дозволити Alexa оцінити енергоспоживання пристрою користувача. Крім того, якщо Alexa відчуває, що

користувач вийшов і забув вимкнути світло, Alexa може автоматично вимкнути його.

Інформаційна панель Alexa Energy Dashboard надає огляд енергії, яку споживають ваші пристрої з підтримкою Alexa, а також персоналізовану інформацію, корисні поради та рекомендовані дії. Інформаційна панель дозволяє користувачам порівнювати тенденції тиждень за тижнем або місяць за місяцем.

В цілому система надає користувачеві можливість контролювати та контролювати обладнання, що використовується. Він також має API для сторонніх розробників для підключення пристроїв до системи. Однак розглянута система не відповідає вимогам програмного продукту. Система в основному призначена для приватних будинків, квартир і т.д.

Найпростіша реалізація даної системи можлива тільки в разі будівництва нового будинку, оскільки має на увазі використання сумісного з нею спеціального обладнання. Таким чином, система не може використовуватися для непідтримуваних пристроїв, і це програмне рішення не має можливості вводити дані вручну.

2.3. CET

CET є провідним постачальником систем енергоменеджменту, пропонуючи комплексні рішення для енергоменеджменту та контролю якості енергії для комерційних, промислових і комунальних ринків.

CET – це продукт, який включає в себе цифрові лічильники потужності та енергії, монітори PQ, шлюзи від Ethernet до RS485, розумні RTU з керуванням і журналюванням Modbus, а також інтелектуальні системи управління енергією, які обслуговують широкий спектр галузей промисловості з 1993 року. Надає економічно ефективні рішення з управління енергією. Розробляє високоякісне цифрове, релейне, мережеве приладдя та інтелектуальні інтегровані системи автоматизації, щоб допомогти створити безпечні, надійні та енергозберігаючі системи живлення. Компанія розробила програмну систему моніторингу PecStar (рис. 2.4). Він розділений на дві підпрограми: програмне забезпечення PecStar iEEM та програмне забезпечення PecStar iEMS.

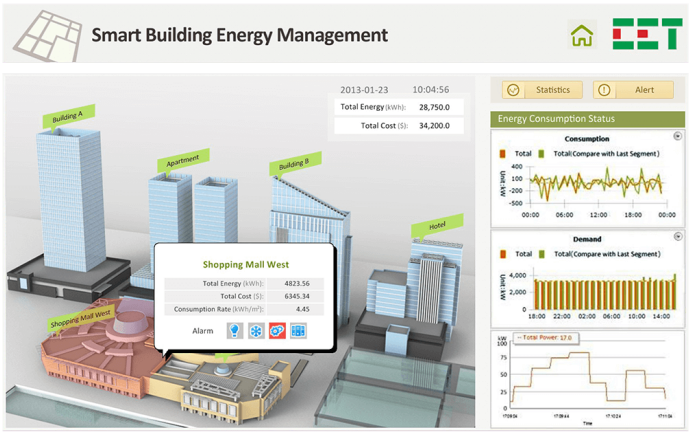


Рисунок 2.4 – ПЗ PecStar

PecStar iEEM — це повна система управління енергоефективністю. Система включає розподілену, модульну, ієрархічну структуру та заходи з управління

енергією, включаючи моніторинг якості енергії для об’єктів, промислових об’єктів, центрів обробки даних і критичних програм.

PecStar iEMS Energy Management System надає комплексне програмне рішення для захисту, моніторингу, контролю та керування електричними, механічними та іншими фізичними активами на об’єктах, галузях, підприємствах, фабриках, комерційних будівлях і центрах обробки даних. Програмне забезпечення також може контролювати пожежі в комерційних будівлях і торгових центрах.

Розглянуте рішення в першу чергу забезпечує моніторинг комерційних підприємств і встановлюється на етапі будівництва будівлі. Ці системи також працюють з лічильниками, виготовленими тією ж компанією, або сумісними пристроями. Як і в інших системах, інформацію неможливо ввести вручну.

Висновки до розділу 2

У цьому розділі було виконано наступне:

1. Проаналізовано аналогічні та подібні системи;
2. Виявлено недоліки та переваги подібних систем;
3. Внесені зміни у розроблюваний додаток;

Аналіз показав, що більшість систем потребують попередньої інсталяції для подальшого використання. Крім того, ці системи підтримують IoT, що робить їх дуже ефективними для збору інформації про споживання

Галузі, в яких можна використовувати ці системи, дуже різноманітні й варіюються від домогосподарств до комерційних організацій. Маймо на увазі, що компанії, які надають програмне забезпечення для моніторингу, зазвичай пропонують власні рішення для збору інформації: розумні лічильники, розумне освітлення та інші пристрої, які інтегруються безпосередньо в їхні системи. Деякі виробники дозволяють використовувати сумісне апаратне забезпечення сторонніх виробників зі своїми системами, що відкриває можливості для реалізації таких систем.

Встановлення цих систем зменшує споживання енергоресурсів, автоматизує систему освітлення та скорочує витрати на облік ресурсів. Ці результати дозволяють судити про те, що такі системи позитивно впливають на енергоспоживання і тому підходять для сучасного світу.

3 АРХІТЕКТУРА МІКРОФРОНТЕНДІВ

3.1. Проблеми монолітів

Більшість фронтендів що розробляються зазвичай мають монолітну архітектуру, тобто є певна команда фронтенд розробників що працюють над одним проектом у рамках одного репозиторія. З самого початку розробки немає жодних проблем у розробці проекту. Кожен розробник має огляд усіх фіч. Фічі створюються швидко. З колегами легко обговорювати теми.

Все це змінюється зі збільшенням обсягу проекту та розміру команди. Як наслідок, розробники перестають розуміти кожного куточка системи. З'являється необхідність у обміні накопиченими знаннями між командою. Складність проекту зростає – зміни в будь-якій частині системи можуть мати непередбачені наслідки. Необхідно робити багато зустрічей щоб тримати всіх у курсі справ. В якийсь момент виникає ситуація, коли додавання нових розробників не підвищує швидкість розробки.

Зазвичай проект ділиться на декілька частин. Такий поділ розділяє програмне забезпечення та структуру команди за технологіями. В більшості випадків відбувається поділ команд на горизонтальні рівні з фронтенд командою та декількома бекенд командами. З плином часу та розростанням проекту до команди додається все більше людей – як на бекенд, так і на фронтенд. У випадку бекенду – мікросервісна архітектура вже стала класикою розробки серверних додатків. З фронтендом ситуація не така однозначна. Розповсюдженим є явище поділу фронт моноліту на умовні частини, наприклад: профіль користувача, головна сторінка, сторінки товару, тощо. Таким чином виділяється команда під кожну частину застосунку.

3.2. Проблема розгортання фронтенд моноліту

Як зазначалося у попередньому розділі великі проекти поділяються на багато команд, що веде за собою необхідність у синхронізації коду та досить частій комунікації між командами, що призводить до зниження ефективності розробки програмного продукту. Однак окрім цих проблем з'являється ще проблема розгортання системи та її доставка до кінцевого користувача.

Розглянемо проблему деплоя монолітного фронтенду який розробляється кількома командами. Нехай це буде команда А та команда Б. Команда А займається сторінкою авторизації, а команда Б – профілем користувача. Обидві команди працюють по Git Flow(рис. 3.1).

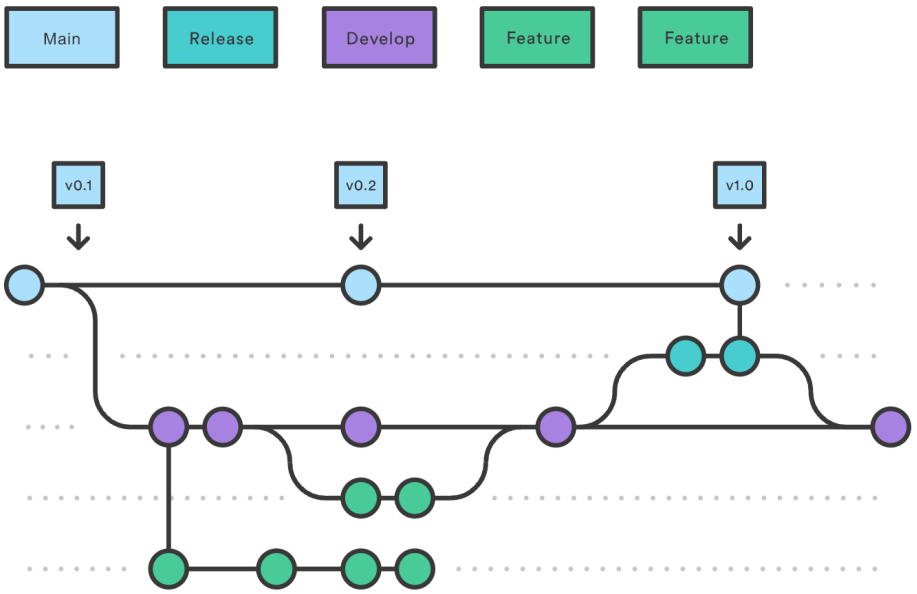


Рисунок 3.1 – Загальний вигляд робочого процесу по Git Flow.

Процес роботи над проектом виглядає наступним чином:

1. Команда бере в роботу певну задачі;
2. Команда починає розробку цієї задачі;
3. Задача розроблена;

4. Відбувається тестування розробленого функціоналу QA командою.
5. Завершення тестування – якщо функціонал працює належним чином – фіча мержиться у develop гілку; якщо ні – задача повертається до розробників на доопрацювання і знову переходить в тест після правок.
6. Після того як код потрапив на develop гілку відбувається проходження паплайну(запуск лінтера, тестів, збірка застосунку) та деплой.

Оскільки маємо один фронтенд репозиторій, то командам необхідно працювати таким чином щоб не перезатиратии зміни одна одної – пильний погляд на роботу іншої команди, оновлення своєї локалної гілки кожного разу коли інша команда щось зливає у головну гілку. Безумовно під час роботи обидві команди стикаються з чималою кількістю merge conflicts. Припустимо, що процес деплою займає 10 хвилин.

Тоді доставка нового функціоналу від двох команд виглядатиме наступним чином: команда що першою завершила свою задачу зливає свій код у головну гілку, починається процес збірки та деплою проекту; інша команда стягує собі зміни попередньої команди та завершує свою задачу. Припустимо що команда А першою завершила свою задачу і зараз її фіча деплоїться, тоді команді Б необхідно дочекатися коли деплой завершиться і запустити новий – зі своїми змінами. Поки ніяких проблем немає, але це тільки лише на перший погляд.

Ситуація змінюється коли у проекті виникають критичні баги, тобто такі які унеможливлюють роботу користувача з системою. Моделюємо ситуацію: у команди А є такий баг – вона негайно починає вирішувати цю проблему, а команда Б працює над фічею яку конче необхідно показати клієнту. Команда Б починає деплоїти свою фічу і тут до неї приходить команда А зі своєю критичною багою. Маємо наступну ситуацію: процес деплою вже запущений, однак необхідно задеплоїти вирішення критичної проблеми. Тоді у команд є декілька шляхів вирішення: зупинення поточного деплою та запуск нового з

вирішенням критичної проблеми, або послідовно все задеплоїти. В першому випадку зміни команди Б не будуть доставлені вчасно та займе, а в другому випадку деплой буде тривати в 2 рази довше, а компанія що володіє цим сайтом потенційно понесе втрати(у випадку інтернет магазину, або схожих платформ). Такі проблеми притаманні великим проектам. Ситуація буде ще складнішою коли кількість команд буде більшою ніж 2(великі продукти мають команди для кожної секції застосунку). Що більше буде команд то більше знадобиться синхронізовуватися командам – вирішувати чиї зміни першими потраплять до користувача. Якщо припустити що всі команди мають критичні баги, то тоді до моменту їх повного вирішення може пройти досить багато часу.

Така проблема у розробці вказує на потребу в ізоляції команд – необхідно позбавитись від постійної синхронізації команд, а з технічного боку зробити доставку функціоналу/правок різних команд автономною, тобто такою що не залежить від інших команд.

3.3. Ідея мікрофронтендів

На відміну від монолітів, мікрофронтенд описує альтернативний підхід. Він ділить додаток на вертикальні фрагменти. Кожен фрагмент створюється від бази даних до інтерфейсу користувача та виконується спеціальною командою. Різні командні інтерфейси інтегруються в браузер клієнта, щоб сформувати остаточну сторінку. Цей підхід пов’язаний з архітектурою мікросервісів. Але головна відмінність полягає в тому, що служба також включає свій інтерфейс користувача. Це розширення сервісу усуває потребу в центральній команді інтерфейсу. Ось три основні причини, чому компанії приймають архітектуру мікрофронтендів:

1. Оптимізація для розробки фіч (задач) — команда включає в себе всі навички, необхідні для розробки функції. Координація між окремими фронтенд і бекенд-командами не потрібна;
2. Спрошення оновлень фронтенду — кожна команда володіє своїм повним стеком від інтерфейсу до бази даних. Команди можуть вирішити оновити або змінити свою технологію зовнішнього інтерфейсу самостійно;
3. Збільшення уваги до клієнта. Кожна команда надає свої функції безпосередньо клієнту. Чистих команд API чи операційних груп не існує.

На рисунку 3.2 наведено огляд усіх частин, важливих для впровадження мікрофронтендів. Мікрофронтендів не є конкретною технологією. Це альтернативний організаційний та архітектурний підхід. Ось чому на цій діаграмі ми бачимо багато різних елементів, як-от структура команди, методи інтеграції та інші пов’язані теми. Все починається з трьох команд над пунктирною лінією і просувається далі вгору. У нижній частині цієї діаграми можна побачити вміст цього вікна у збільшеному масштабі. Воно ілюструє три різні аспекти: роутинг, композиція компонентів системи, комунікація між компонентами застосунку. Ці аспекти необхідні для створення інтегрованого

застосунку.

Три коробки з командами A(Team A), B(Team B) і C(Team C) демонструють вертикально розташовані програмні системи. Вони складають ядро цієї архітектури. Кожна система є автономною, що означає, що вона може працювати, навіть коли сусідні системи не працюють. Для цього кожна система має власне сховище даних. Крім того, для відповіді на запит він не покладається на синхронні виклики інших систем.

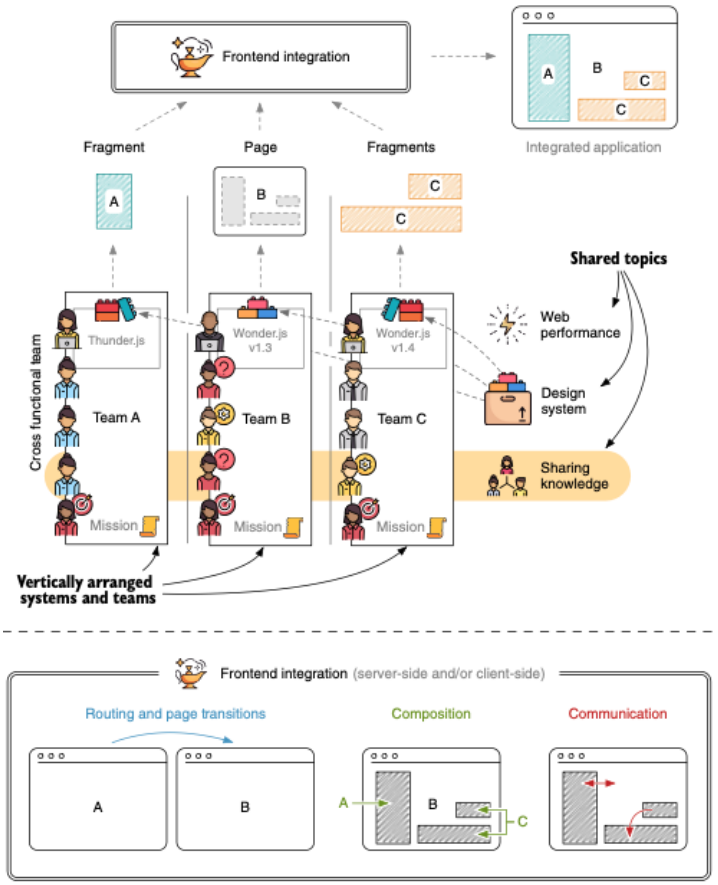


Рисунок 3.2 – Загальний огляд архітектури мікрофронтендів.

Однією системою володіє одна команда. Ця команда працює над усім пакетом програмного забезпечення зверху вниз. Кожна команда має свою сферу знань та область відповідальності, у якій вона забезпечує цінність для клієнта.

3.4. Основні патерни для побудови мікрофронтендів

Існує два взаємодоповнюючі методи візуалізації уніфікованого інтерфейсу користувача з окремих компонентів мікрофронтендів: рендеринг на стороні сервера(англ. Server side) та на стороні клієнта(англ. client side).

Візуалізація на стороні сервера забезпечує швидшу продуктивність і більш доступний вміст. Візуалізація на сервері є хорошою альтернативою для швидкого завантаження вмісту, особливо на пристроях із низьким енергоспоживанням, як-от недорогі телефони. Це також придатний резервний режим, коли JavaScript вимкнено (рис. 3.3).

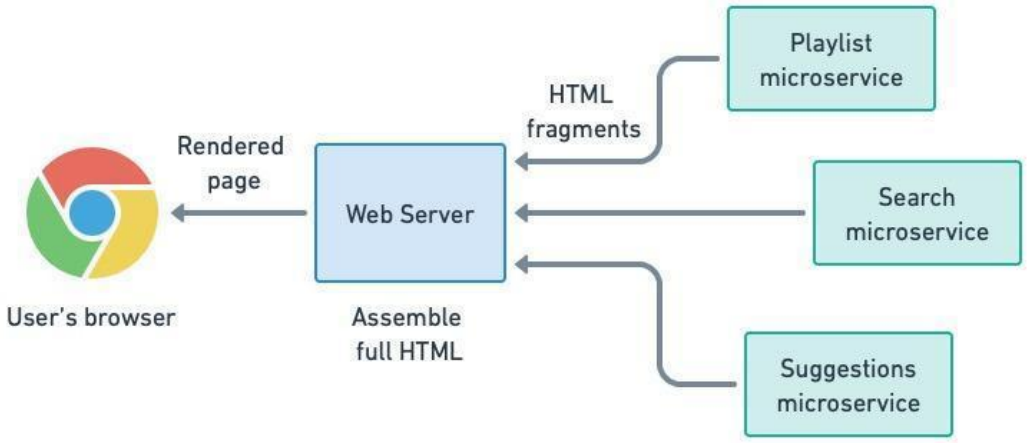


Рисунок 3.3 – Рендеринг на стороні сервера.

Веб-сервер збирає повну сторінку з вмісту, наданого різними мікросервісами. Це перша сторінка зі змістом. Коли вміст завантажився клієнту JavaScript можна використовувати для додавання більш динамічного вмісту. Наступні методи можна використати для побудови SSR:

- Server Side Includes (SSI): це проста скриптова мова, яка виконується веб-сервером. Мова використовує директиви для побудови фрагментів HTML у повну сторінку. Ці фрагменти можуть походити з інших файлів

або відповідей програм. SSI підтримується всіма основними веб-серверами, включаючи Apache, Nginx і IIS;

- Iframes – функція iframe дозволяє нам вставляти довільний вміст HTML на сторінці;
- Edge Side Includes (ESI) – більш сучасна альтернатива SSI. ESI може обробляти змінні, мати умови та підтримує кращу обробку помилок. ESI підтримується кешуванням HTTP-серверів, таких як Varnish.

SSR використовується у сучасних фреймворках для того щоб відобразити сторінку якомого швидше – в цьому його перевага над клієнтським рендерингом.

Рендеринг на стороні клієнта(англ. Client Side Rendering або CSR) створює сторінку в браузері користувача шляхом отримання даних із мікросервісів(мікрофронтендів) і маніпулювання DOM (рис. 3.4). Більшість веб-фреймворків використовують певну форму CSR для покращення взаємодії з користувачем.

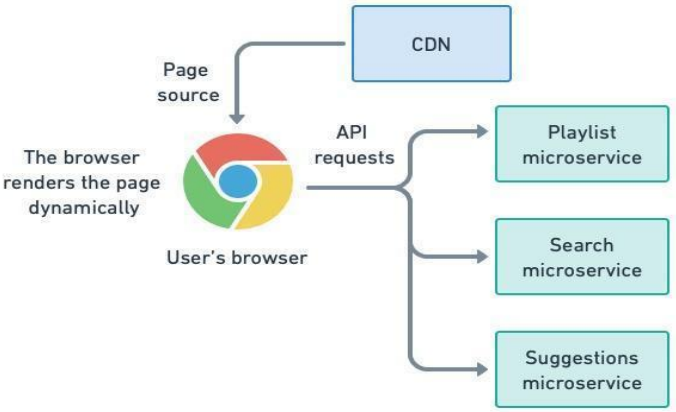


Рисунок 3.4 – Рендеринг на стороні клієнта(браузера)

CSR динамічно відображає сторінку в браузері користувача, використовуючи дані, надані кінцевими точками. Основними інструментами, які ми маємо для

написання слабозв’язаних компонентів, є власні елементи. Спеціальні елементи(Custom Elements) є частиною стандарту HTML. Вони дозволяють нам створювати нові теги HTML і додавати до них логіку та поведінку. Хоча більшість інтерфейсних фреймворків можна використовувати для мікрофронтендів, деякі з них були розроблені спеціально для них:

- Piral: реалізує ізольовані компоненти, які називаються пілетами. Пілети — це модулі, які об’єднують вміст і поведінку;
- Ragu: каркас каркасів. Це дозволяє вбудовувати код, написаний у будь-яку структуру, як віджети;
- Single SPA: метафреймворк для об’єднання інтерфейсу користувача за допомогою будь-якої комбінації фрнтенд фреймворків, таких як React, Angular і Ember тощо;
- WebPack Module Federation: плагін WebPack для створення односторінкових програм (SPA) шляхом об’єднання окремих збірок. Ці збірки можна розробляти незалежно одна від одної.

3.5. Слабкі та сильні сторони мікрофронтендів

Як бачимо мікрофронтенти мають багато переваг над звичайним монолітним підходом, однак вони не позбавлені своїх недоліків. Спочатку перерахуємо сильні сторони цього підходу:

- Краща масштабованість;
- Швидка розробка, оскільки команди можуть працювати незалежно. Ця перевага, однак, застосовна, лише якщо проект великий і має більше ніж одну фронтенд команду;
- Можливість використовувати різні технології та фреймворки. Однак з цією можливістю слід бути обережним, тому що частіше всього різні фреймворки ускладнять проект та можуть призвести до неочікуваних наслідків.
- Незалежність розгортання. Доставка мікроінтерфейсу не вплине на всю програму. Зміни торкнуться саме тієї частини бізнес-процесу, яку вони охопили;
- За допомогою мікрофронтендів можна оновити, або навіть переписати частини інтерфейсу більш плавно, ніж це можливо з монолітом;
- Легше переконатися, що решта програми залишається стабільною, оскільки вона незалежна. З мікрофронтендами більше не потрібно перевіряти весь додаток. Команда перевіряє певну частину, яку було змінено або розширено;
- Коду стає менше та він легший для керування;
- Простіше тестування, оскільки тестуються лише окремі функції.

Як не дивно, з сильних сторін можуть виникнути і слабкі, наприклад можливість застосовувати різні фреймворки та їх версії в результаті збільшує кількість файлів що треба завантажити клієнту, що значно впливає на швидкість роботи. сиситеми. Перелічимо основні недоліки мікрофронтендів:

- Комплексне тестування програми в цілому. Тепер, програма динамічно завантажує вміст, може бути важче мати повне уявлення про програму. Кожен зовнішній інтерфейс можна перевірити окремо, але тестування реального користувача є критичним для забезпечення роботи програми для кінцевого користувача;
- Велика різноманітність стандартів, яких треба дотримуватися. Додаток розбивається на менші частини; таким чином, може бути складно змусити всіх розробників працювати за однаковими стандартами. Є необхідність у постійному обміні знаннями між розробниками для забезпечення високоякісної взаємодії з користувачем;
- Застосування мікрофронтендів — гарна ідея для бізнесу, якщо проект великий і має більше ніж одну фронтенд команду. З невеликою командою доцільність такого підходу знаходиться під питанням;
- Процес деплою, збірки та конфігурації для кожного мікрофронтенду може бути різним, що вимагає додаткових зусиль.

Висновки до розділу 3

Даний розділ містить багато теоретичної інформації. У розділі було описано проблеми що виникають у розробці клієнтської частини, наведено приклад одної з найскладніших проблем монолітів; вказано на недоліки монолітного підходу. Також було розглянуто основну ідею мікрофронтендів як альтернативний підхід до розробки, розглянуто способи досягнення мікрофронтендної архітектури. Розглянуто патерни для побудову мікрофронтендів та наведено схеми що пояснюють відмінність між підходами. В кінці розділ містить детальний опис слабких та сильних сторін такого підходу.

4 ВИБІР ТЕХНОЛОГІЙ ДЛЯ ВЕБ-ЗАСТОСУНКУ

4.1. TypeScript

TypeScript — це мова з відкритим кодом на основі JavaScript. TypeScript розвинувся з багатьох недоліків JavaScript у широкомасштабну реалізацію Microsoft та іншими користувачами JavaScript. Зауважимо, що синтаксис TypeScript дещо схожий на синтаксис C#, оскільки основним розробником TypeScript був Андерс Хейлсберг, який зробив внесок у розвиток мови C#. Проблеми з підтримкою складних програм JavaScript створили потребу в спрощенні розробки компонентів мови.

Насправді ця мова є мовою JavaScript. Оскільки після створення проекту, написаного цією мовою, TypeScript компілюється в JavaScript, який потім виконується компілятором. TypeScript має значну перевагу перед JavaScript:

- Анотації;
- Модифікатори доступу(public, private, protected);
- Інтерфейси;
- Узагальнене програмування;
- Абстрактні класи;
- Інструменти для наслідування класів;
- Mixins;
- Типи;
- Enum.

Увесь код JavaScript також є кодом TypeScript. Однак спроба використовувати код JavaScript як TypeScript може призвести до помилок перевірки типу, але не завадить виконанню отриманого JavaScript. Код TypeScript перетворюється на код JavaScript за допомогою компілятора TypeScript або Babel. Отриманий

скомпільований код JavaScript є чистим і простим кодом, який працює на JavaScript (браузер, Node.JS або інші програми).Мова підтримується багатьма редакторами коду та IDE такими якVisual Studio Code, Nova, Atom, Sublime Text, Vim та іншими. На основі TypeScript написано багато ПЗ із відкритим кодом, серед яких:

- Angular;
- Vue;
- Засіб для тестування Jest;
- Бібліотека для unit тестів Jasmine;
- Ionic;
- NestJs;
- NextJs;
- React;
- Застосунок GitHub.
- Slack;

І це ще не повний перелік технологій що використовують дану мову. Деякі програми було спеціально переписано з JavaScript на TypeScript для забезпечення більш високої якості коду. Таким чином розробники отримали меншу кількість багів(помилок), кращий процес розробки, забезпечення контрактів що використовує система, бо типи що використовуються на бекенді синхронізовані з тими що використовує фронтенд.

4.2. Вибір фремворка

Наразі існує досить багато веб-фреймворків які використовують для розробки веб-застосунків, серед яких слід виділити наступні (рис. 3.1):

- Angular;
- React;
- Vue;
- NextJs;
- Svetle.

Рисунок 4.1 – Популярні фронтенд фереймворки

Чистий JavaScript, HTML і CSS використовуються рідко. Винятком є сайти-візитки або односторінкові сайти, які не пропонують багато функцій (редагування даних, ліцензування тощо). Фреймворк значно прискорює розробку програмного забезпечення, надаючи готові інструменти, такі як вбудована маршрутизація, рендерери об’єктів, клієнти для HTTP-запитів та інші корисні інструменти. SPA застосунок став стандартом який і реалізовує більшість фреймворків. Таблиця 4.1 демонструє використання різних фреймворків компаніями для створення та підтримки своїх програмних продуктів.

Таблиця 4.1 – Фремворки та побудовані на їх основі застосунки.

Фреймворк	Застосунок
React Framework	• Dropbox • Instagram • Airbnb • Discord • Pinterest
Angular Framework	• Microsoft Office • Deutsche Bank • Mixer • Santander • Gmail • Forbes
Фреймворк Vue.js	• FindlayWebTech • Adobe • BridgeU • Gitlab
Ember.js	• Twitch • LinkedIn • Accenture • Square • DigitalOcean
Svelte Framework	• Capital One • Snap! Raise • GoGuardian

Як бачимо, досить багато застосунків якими ми користуємось увесь час збудовані саме на осноні фреймворків. Можна сказати що вибір фреймворку є відповідальною частиною при початку розробки будь-якого програмного продукту.

Очевидно що поміж усіх існуючих рішень, обрати саме те що потрібно досить складно. По-суті усі вони виконують одну й ту саму роботу – допомагають збудувати веб-додаток. У випадку цієї роботи вибір основного фреймворка є досить очевидним – це фреймворк Angular п’ятнадцятої версії.

Такий вибір обумовлений тим що дана робота є переписом зі значними покращеннями бакалаврської роботи “Кабінет енергоменеджера” яка була написана на такому ж самому фреймворку. Однак, це не єдина причина чому було обрано саме такий фреймворк. З 15 версії фремворк став значно швидше – новий механізм рендеренгу Ivy, підтримка типізованих форм, спрощення роботи з компонентами – Standalone Components. Також цей фремворк вважається одним з кращих для написання великих корпоративних додатків. Відзначимо що фреймворк має підтримку архітектури мікрофронтендів через відповідні пакети.

4.3. RxJS

RxJS — це інтерактивна бібліотека програмування, яка дозволяє легко писати асинхронний або зворотний код за допомогою Observables. Цей проект є реалізацією Reactive Extensions для JavaScript/TypeScript (рис. 3.2), зберігаючи більшу частину зворотної сумісності з покращенням продуктивності, модульності та налагодження стеку викликів.

Рисунок 4.2 – Бібліотека RxJs

ReactiveX або Reactive-Extensions є поєднанням найкращих ідей стилю спостерігача, рекурсивного стилю та функціонального програмування. RxJ складається з таких сутностей:

- Operators;
- Observables;
- Schedulers.

У RxJs спостережувані підписуються на спостережувані. Потім цей спостерігач реагує на дію або серію дій, які виконує спостерігач. Цей стиль полегшує паралельну роботу, оскільки потоки спостерігачів не повинні блокувати, чекаючи сповіщення спостерігача про деякі зміни. Натомість підготуйте спостерігача відповідати належним чином щоразу, коли про це

попросить спостережлива особа. Тип Observable додає відсутню семантику до оригінального патерну “групи з чотирьох” Observer, щоб відповідати семантиці, наданій ітерованим типом:

1. Здатність виробника повідомити споживача про те, що більше немає даних (у цьому випадку цикл foreach Iterable завершився успішно; Observable викликає метод onCompleted в Observer).
2. Завдання виробника — повідомити споживача про те, що сталася помилка (якщо помилка виникає під час ітерації, ітератор видає виняток (помилку); Observable викликає метод onError в Observer).

Завдяки цим доповненням ReactiveX поєднує відтворювані та видимі стилі. Єдина відмінність між ними полягає в тому, у якому напрямку передаються дані. Це дуже важливо, тому що будь-яка операція, яка може бути виконана на Iterable, також може бути виконана на Observable.

Observer та Observable – це основа ReactiveX. Самі по собі вони представляють реалізацію стандартного шаблону спостерігача, який краще підходить для обробки послідовності подій. Одним із найпотужніших інструментів у ReactiveX або RxJ є оператори, які дозволяють трансформувати, комбінувати та маніпулювати послідовністю об’єктів, створених спостережуваними. Перевага цих операторів полягає в тому, що ви можете декларативно скомпілювати асинхронні послідовності з усіма перевагами ефективності зворотних викликів, але без недоліків вкладених обробників зворотних викликів, тому ваш код чистіший і легший для розуміння. Оператори RxJ класифікуються таким чином:

- Створення нових Observable;
- Трансформація потоку;
- Фільтрація;
- Комбінування;
- Обробка помилок;

- Умова;
- Агрегація/Конвертація.

Дуже зрозумілий приклад роботи операторів RxJs дає оператор filter, дію якого відображено на рисунку 4.3. Маємо наступну картину – вхідний потік, та вихідний потік що утворюється внаслідок роботи оператора.

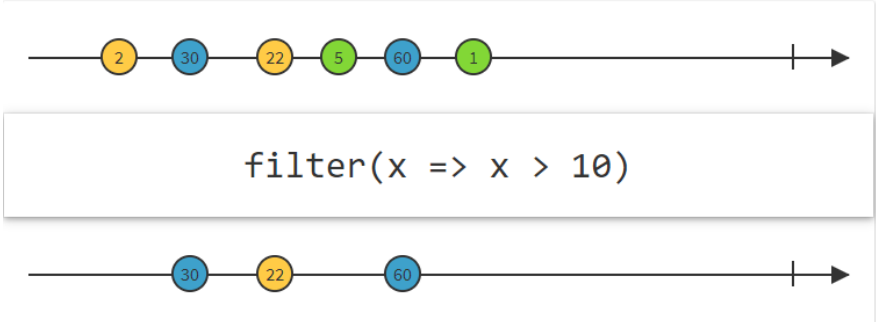


Рисунок 4.3 – Дія оператора filter

Оператор фільтрує значення, що надходять у вхідний потік, на основі якогось стану. У цьому випадку умовою є перевірка того, чи значення елемента більше 10. Таким чином, цей оператор дозволяє лише елементам, які пройшли заданий тест, бути увійти до потоку після цього оператору. Всі інші оператори працюють так само. Тобто основною функцією оператора є зміна вхідного потоку.

4.4. Ngrx як реалізація шаблону Redux

Оскільки вимоги односторінкових програм JavaScript з часом ускладнюються, вам потрібен більш детальний контроль над станом програми. Стан програми являє собою інформацію що отримана від сервера та кеш, а також дані необхідні для роботи самої програми або дані що ввів користувач, які ще не збережено на сервері. Стан GUI також складний, оскільки він має керувати активним шляхом, вибраною вкладкою, індикаторами завантаження, елементами керування сторінкою тощо. У якийсь момент ви втрачаєте контроль над тим, коли, чому та як змінюється стан вашої програми, що ускладнює визначення того, що відбувається у вашій програмі. Важко відтворювати помилки або додавати нові функції, коли система непрозора та недетермінована.

Для організації стану програми було створено шаблон Redux. Шаблон Redux поєднує такі шаблони дизайну:

- Flux(потік) – схема керування потоком даних, яку можна використовувати в застосунку. Рух даних відбувається в одному напрямку.
- CQRS (Command Query Responsibility Separation) — можливість оновлювати інформацію за допомогою команд що відповідають за події зміну стану. Event Sourcing – послідовність подій що змінює стейт.

Redux передбачає зміни стану, накладаючи деякі обмеження на те, як і коли відбуваються оновлення. Ці обмеження відображені в трьох принципах Redux:

- Стейт тільки для зчитування;
- Одтин стейт(стан) відображає весь стан системи;
- Чистих функції як інструмент зміни стейту.

Redux має лише одну сутність, яка утримує (зберігає) стан програми. Цей метод полегшує налагодження та тестування програми. Єдиний спосіб змінити стан програми — опублікувати процедуру, об'єкт, який описує те, що сталося. Чисті функції, які називаються редукторами(reducer), використовуються для визначення того, як дії змінюють дерево станів.

Бібліотека NgRx є реалізацією шаблону Redux, але містить низку розширень, які значно покращують початковий шаблон. Однією з найважливіших особливостей цієї бібліотеки є використання rxjs як представлення даних. Це робить бібліотеку більш сумісною з фреймворками Angular, які використовують RxJ як систему керування станом. Можна сказати, що ця бібліотека в основному створювалася для цього фреймворку. Взаємодія компонентів бібліотеки показано на рисунку 4.4.

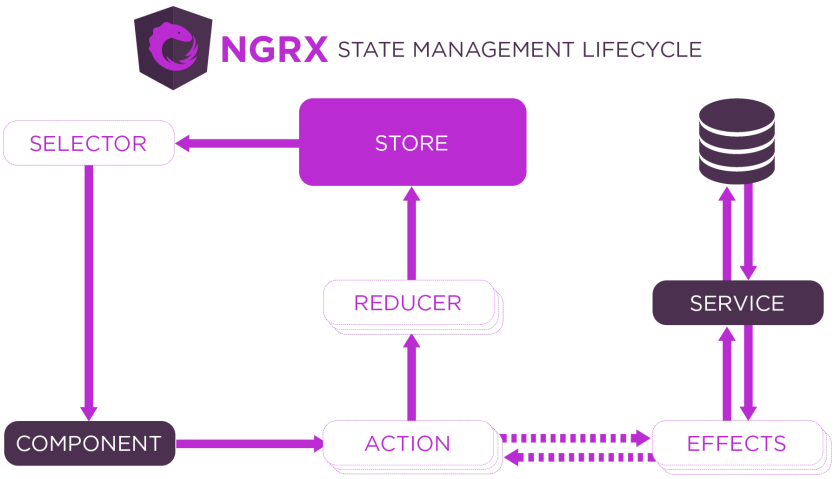


Рисунок 4.4 – NgRx у Angular застосунку

У порівнянні з Redux, бібліотека поєднує дві сутності: селектори та ефекти. Селектори використовуються для вибору певних даних зі сховища. Сутності впливу(Effects) – це класи, які не виконують зовнішніх дій, пов'язаних із змінами збереженого стану. Прикладами таких атрибутів є http-запити, математичні обчислення, виклики методів обслуговування тощо.

4.5. Ngx-Bootstrap

Для прискорення розробки компонентів програми використовувалася бібліотека ngx-bootstrap. Ця бібліотека є розширенням бібліотеки Bootstrap, яка розроблена для використання з Angular Framework. Бібліотека містить готові до використання рішення для багатьох графічних елементів, таких як:

- Accordeon;
- Warning;
- Buttons;
- Slider;
- DatePicker;
- Dropdowns;
- Modals;
- Hints;
- Spinners;
- TimePicker.

Усі вони мають власні API, які дозволяють контролювати ці компоненти. Кожен компонент можна додатково налаштувати відповідно до дизайну. Крім того, ця бібліотека постачається з набором глобальних стилів, налаштованих для прискорення обробки розмітки сторінок.

4.6. Менеджер пакетів NPM

Під час створення проекту, використовуються різні інструменти, щоб полегшити та пришвидшити розробку. У більшості випадків ці інструменти створені іншими розробниками та опубліковані для безкоштовного використання.

NPM(акронім node package manager) — це менеджер пакетів для проектів Node.js, доступний для загального використання. Проекти, доступні в реєстрі npm, називаються «пакетами». NPM дозволяє легко використовувати код, написаний іншими, без необхідності писати його самим під час розробки. Реєстр npm містить понад 1,3 мільйона пакетів, якими користуються понад 11 мільйонів розробників у всьому світі. Це менеджер пакетів може використовуватися для наступного:

- Адаптація пакетів коду для програм, або включення пакетів;
- Завантаження інструментів;
- Спільний доступ до коду з будь-яким користувачем npm будь-де;
- Обмеження доступу до коду для певних розробників;
- Оновлювати застосунки коли бібліотеки від яких вони залежать отримують оновлення;
- Створення та викладання власних бібліотек та пакетів.

Висновки до розділу 4

У цьому розділі описано вибраний стек технологій програми. Крім того, кожен пункт містить опис обраної технології та пояснення такого вибору. Підбиваючи підсумки можна сказати що:

1. Було обрано основний фреймворк для розробки застосунку;
2. Передбачено управління станом застосунку;
3. Обрано відповідні бібліотеки що забезпечують швидку розробку;
4. Використано менеджер пакетів для встановлення бібліотек.

.

5 ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ

Розробка системи з таким широким спектром можливостей вимагає хороших інструментів розробки. Інструмент, який ви виберете, має бути простим у використанні та прискорювати процес розробки. Щоб створити веб-додаток, необхідно вибрати такі інструменти:

- Платформи або програми, які дозволяють переглядати загальний дизайн системи, основну колірну схему програми, базові відступи елементів, а також дозволяють завантажувати зображення, які можна використовувати для створення інтерфейсу (наприклад, фонові зображення для піктограм), тощо;
- Інтегроване середовище розробки (IDE) – програма або набір програмного забезпечення для розробки програмного забезпечення, що включає текстові редактори, налагоджувачі та засоби інтерпретації/компіляції коду;
- Переглядач системного API, що має перелік усіх публічних ендпоінтів. Має можливість показати моделі що приймаються як вхідні параметри та моделі що слугують вихідними параметрами.

5.1. Figma

Щоб створити веб-додаток, необхідно спочатку створити макет сайту, а потім додати бізнес-логіку до елементів сайту. Тому, щоб прискорити процес планування, варто скористатися спеціалізованою програмою, яка показує весь дизайн і містить основні примітки по стилю (style guide). В якості програми обрано додаток Figma (рис. 5.1). Додаток також дозволяє створювати прототипи сайтів, розробляти інтерфейс користувача та організовувати співпрацю в режимі реального часу.

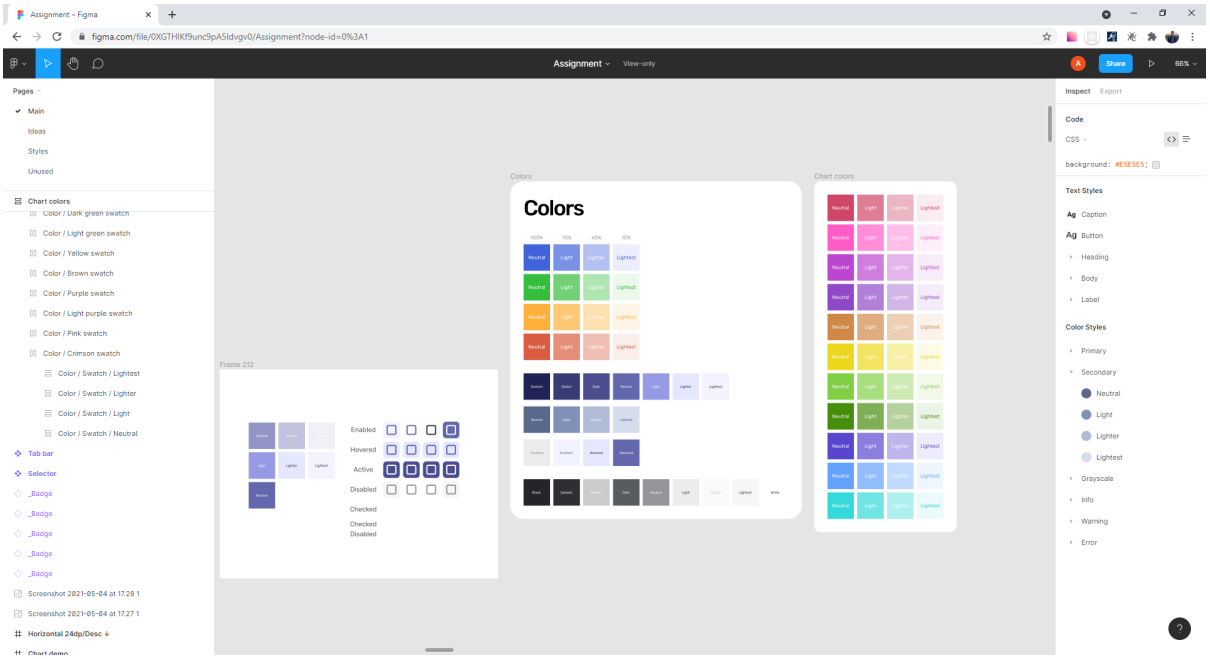


Рисунок 5.1 – Платформа Figma

Застосунок дозволяє переглядати та редагувати ваші проекти в реальному часі, тож ви можете змінювати макети на льоту. Ця програма також є редактором SVG, який є дуже потужним редактором векторної графіки. Платформа складається з настільних і веб-додатків, які синхронізуються між собою. Деякі з

найважливіших функцій Figma з точки зору компонування:

- Експорту іконок у форматі SVG (рис. 5.2);
- Генерація стилів CSS, з властивостями та змінними які використовуються у цьому елементі (рис. 5.3);
- Розміри елементу;
- Інтерактивність між дизайнером та розробниками що досягається за рахунок коментарів, які можна поставити на елемент дизайну для подальшого обговорення.

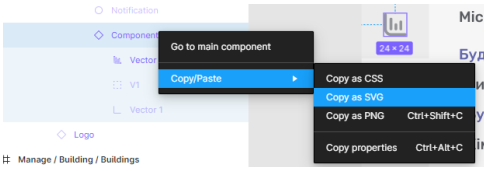


Рисунок 5.2 – Експорт іконки як SVG

Рисунок 5.3 – Підказки до стилів

Як бачимо, Figma значно переважає перед такими програмами, як Photoshop та Illustrator, оскільки не вимагає ліцензії користувача та має простий у використанні інтерфейс для розробників. Можна сказати що дане програмне рішення вже встигло стати одним зі стандартних інструментів для розробників та дизайнерів.

5.2. WebStorm як основний засіб розробки

Для створення самої веб-програми будемо використовувати IDE що має назву WebStorm від виробників JetBrains (рис. 4.4). Звернемо увагу, що програма є платною, але якщо ви студент, розробник пропонує безкоштовну ліцензію, яку ви можете отримати зі студентським квитком. Альтернативою Webstorm є Visual Studio Code, але він не такий простий у використанні, як Webstorm. Однак, на відміну від Webstorm, Visual Studio Code абсолютно безкоштовний.

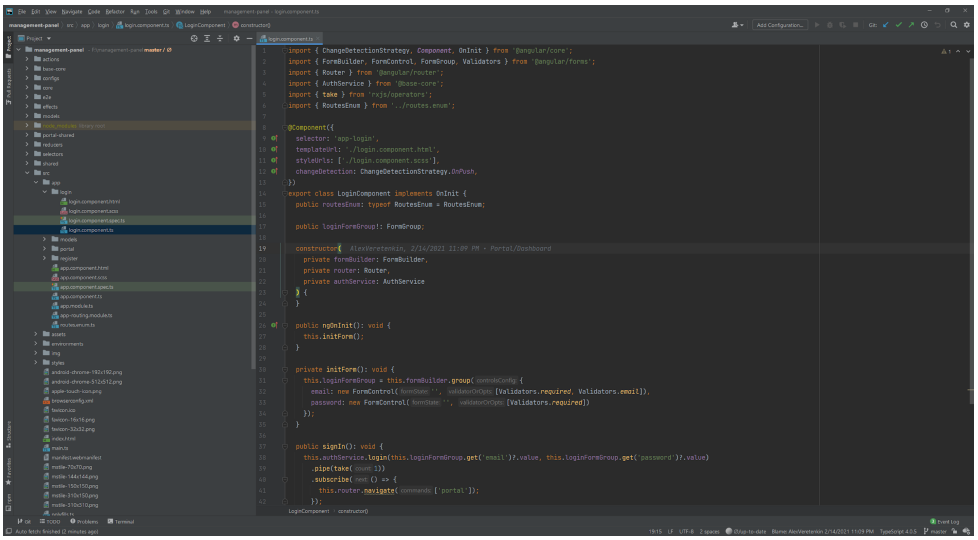


Рисунок 5.4 – IDE WebStorm

IDE дозволяє виконувати пошук по всьому проекту, генерувати класи, інтерфейси. Редактор підсвічує ключові слова, модифікатори доступу, зміні та інші елементи програми. Дана програма має свій набір розширень які встановлюються у вкладці “marketplace” – ці розширення дозволяють додати підтримку нових мов, фреймворків, або, наприклад, змінити зовнішній вигляд застосунку.

5.3. OpenApi

Специфікація OpenAPI (колишня специфікація Swagger) була передана Linux Foundation у 2015 році в рамках ініціативи OpenAPI. Ця специфікація створює інтерфейс RESTful, який полегшує розробку та використання API і ефективно відображає всі задіяні ресурси та процеси. Діаграму взаємодії зображено на рисунку 5.5.

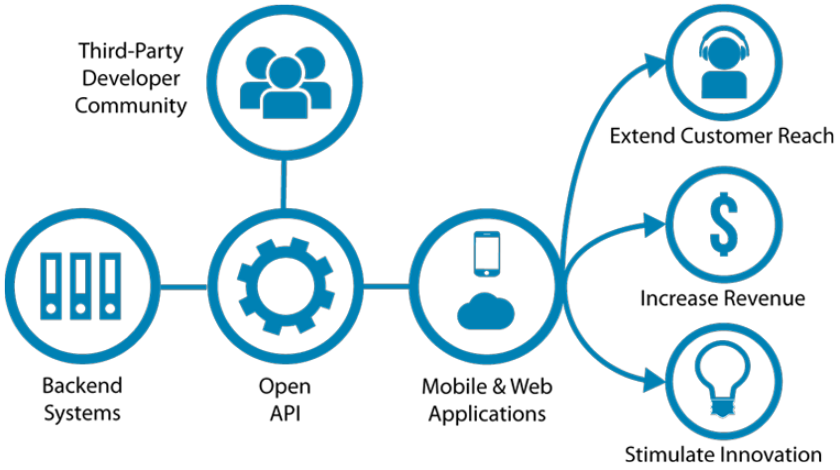


Рисунок 5.5 – OpenAPI взаємодіє з системою

OpenAPI надає можливість описати API, що включає в себе:

- Ендпоінти(/PROFILES) і операції що до них застосовуються;
- Моделі параметрів;
- Авторизація;

Формат, необхідний для написання специфікації - YAML або JSON.

5.4. Swagger

Swagger — це набір інструментів з відкритим кодом, заснований на специфікації OpenAPI. Swagger допомагає виконувати проектування, створення, документування та використання REST API. Ключові інструменти Swagger:

- Редактор що описує специфікацію OpenAPI;
- Інтерактивна API документація (рис. 5.6);
- Генерація коду(codegen).

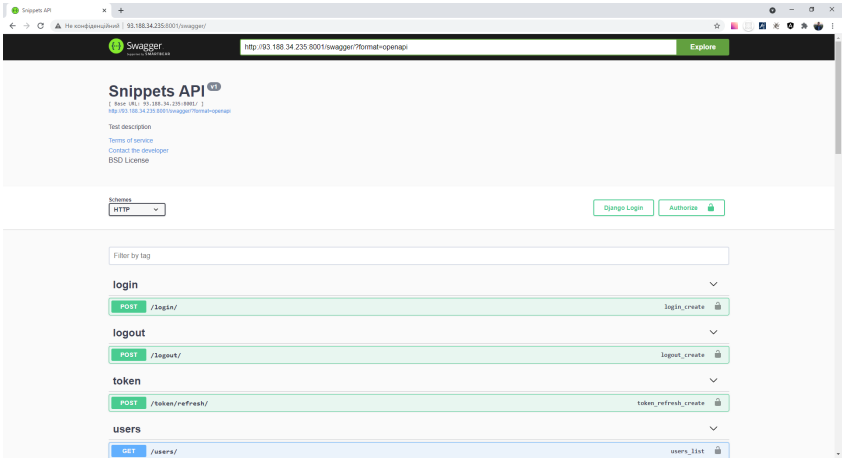


Рисунок 5.6 – Список доступних ендпоінтів.

З точки зору розробників веб-додатків, найважливішим інструментом Swagger є інтерактивна API документація. Використовуючи цей інструмент, маємо можливість переглядати всі кінцеві точки, доступні у API, надсилати запити на тестувати конкретні кінцеві точок, перевіряти авторизацію та підтримувати роботу систем моделі (класу). Отже, перед виконанням запиту на інтерфейсі ми можемо перевірити запит, проаналізувавши необхідні дані для того, як він працює. Крім того, цей інструмент спрощує процес переміщення класів, які діють як дані вводу/виводу, до інтерфейсу.

Висновки до розділу 5

Даний розділ розглядає різні інструменти для розробки ПЗ. Спочатку було встановлення перелік інструментів за допомогою яких буде здійснено розробку веб-додатка, а потім були обрані наступні інструменти:

1. IDE WebStorm
2. Figma – перегляд дизайну.
3. Swagger та OpenAPI як метод перегляду документації ендпоінтів.

Такий набір інструментів становить необхідний мінімум для розробки будь-якого фронтенд додатку.

6 РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ

6.1. Загальна структура

Проект побудований відповідно до архітектури мікрофронтендів, а це означає що він має містити декілька застосунків що утворюють кінцеву програму. Було вирішено досягти цього шляхом створення монорепозиторія. Таке рішення було обумовлене тип що, по-факту, над застосунком працювала 1 людина і розбиття проекту на різні репозиторії знизило б ефективність та швидкість розробки (рис. 6.1).

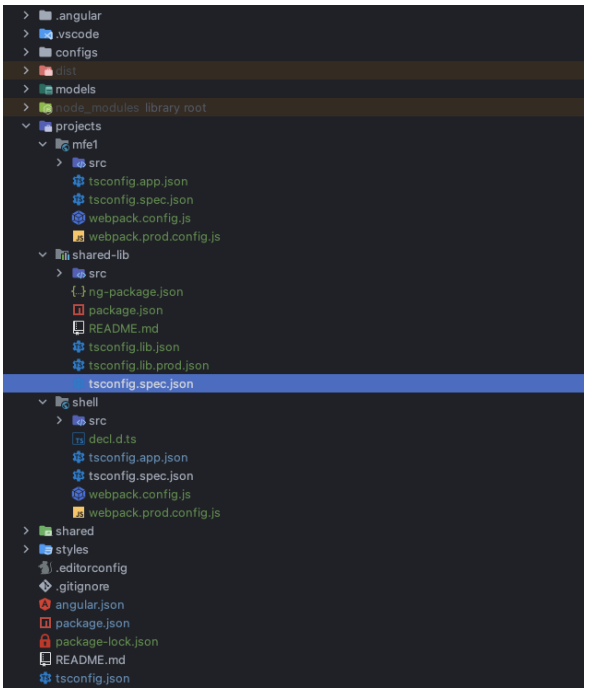


Рисунок 6.1 – Структура монорепозиторія

Як бачимо з рисунку 6.1, застосунок поділяється на проекти, які і є представленням мікрофронтендів. Окрім них у головній теці містяться файли конфігурації та ресурки, необхідні для роботи програми.

Розглянемо основні файли та папки системи:

- projects – містить всі мікрофронтенди та власні бібліотеки;
- mfe1– мікрофронтенд що відповідає за портал застосунку;
- shell – вхідна точка для встого додатку(головний мікрофронтенд);
- shared-lib – бібліотека компонентів та утиліт що доступка усім мікрофронтендам
- node_modules – усі пакети що використовуються у застосунку;
- shared – загалні утиліти;
- styles – глобальні стилі застосунку;
- models – моделі для роботи програми.

Розглянемо структуру окремого мікрофронтенда, вона складається з:

- src – основний код
- assets – ресурси, такі як картинки, конфігурації чи переклади;
- environments – змінні що описують середовище запуску застосунку;
- styles – стилі для конкретного мікрофронтенду;
- img – зображення які використовуються у додатку;
- webpack.config.js – конфігурація налаштування мікрофронтенда;
- main.ts – налаштування запуску.

Такий набір файлів є типовим для загального налаштування мікрофронтенду, однак файлів та папок може бути більше. Наприклад, більш складні мікрофронтенди зі своїм станом та реалізацією значної частини бізнес логіки будуть включати в себе наступні модулі:

- core/ base-core – реалізація бізнес логіки або запитів
- shared – спільні компоненти для всіх модулів мікрофронтенду
- effects – ефекти для роботи з NgRx
- reducers – редуктори для роботи з NgRx
- actions – дії що викликають зміну стану програми
- selectors – вибір даних з глобального сховища(Стану)

6.2 Мікрофронтенд Shell: base-core модуль

Модулі з приставкою “core” являються основними модулями, що служать контейнером для служб, які реалізують бізнес-логіку, наприклад, необхідну для авторизації та перевірки (перевірка правильності введення даних). Цю сутність можна використовувати в будь-якій точці вашої програми(в нашому випадку мікрофронтенду). На рисунку 6.2 показано взаємозв’яок підрозділів і служб.

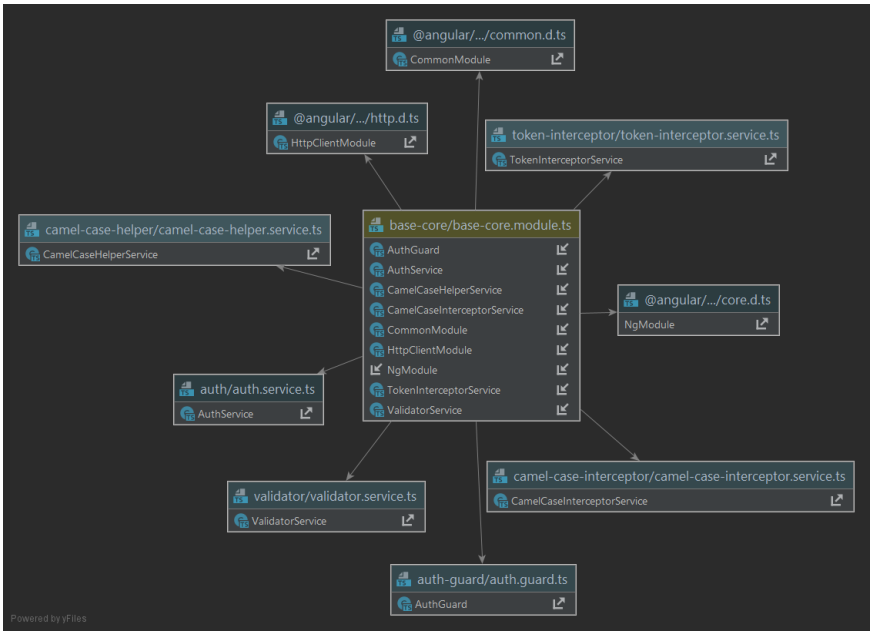


Рисунок 6.2 – Зв’язки base-core

Наступні сервіси утворюють цей модуль:

- AuthService;
- AuthGuardService;
- ValidatorService;
- CamelCaseInterceptorService;
- TokenInterceptorService;
- CamelCaseHelperService.

Зауважимо, що в додатках Angular бізнес-логіка зосереджена в класах, які представляють сервіси. Бізнес-логіка включає математичні обчислення, http-запити, збереження даних у локальному сховищі (local storage), обробку відповіді на http-запит і так далі. Служба(сервіс) авторизації (AuthService) відповідає за вхід і реєстрацію користувача. Крім того, служба керує локальним сховищем браузера для забезпечення авторизації. Тобто там зберігається токен користувача.

Angular має спеціальний тип сервісів – сторожові сервіси(Guard) служби. Цей тип сервісу надає певний бар'єр, який не дозволяє користувачеві отримати доступ до зазначеного ресурсу (URL), якщо певні умови не виконуються. Для служби AuthGuard авторизація є умовою, за якої користувач може отримати доступ до порталу. Це запобігає доступу неавторизованих користувачів до порталу. На додаток до служб безпеки, Angular має своєрідний сервіс, який може взаємодіяти з HTTP-запитами, перехоплюючи їх і обробляючи далі. Ці служби називаються перехоплювачами(interceptors). У цьому модулі є два сервіси. Сервіс, відповідальний за обробку токенів користувача - TokenInterceptor, і сервіс, який транслює вхідні та вихідні дані http-запитів - CamelCaseInterceptor. Служба перетворення даних введення-виведення розроблена таким чином, щоб не порушувати правила іменування. API, написані мовою Python, мають правила іменування регістру snake_case, тоді як TypeScript використовує регістр camelCase.

6.3 Бібліотека Shared-lib: shared модуль

Для того, аби дотримуватися одного стилю та дизайну в усіх частинах застосунку зазвичай створюються свої бібліотеки з компонентами що слугують будівельними блоками для побудови застосунку. Такою бібліотекою є shared-lib, а саме її модуль shared (рис. 6.3):

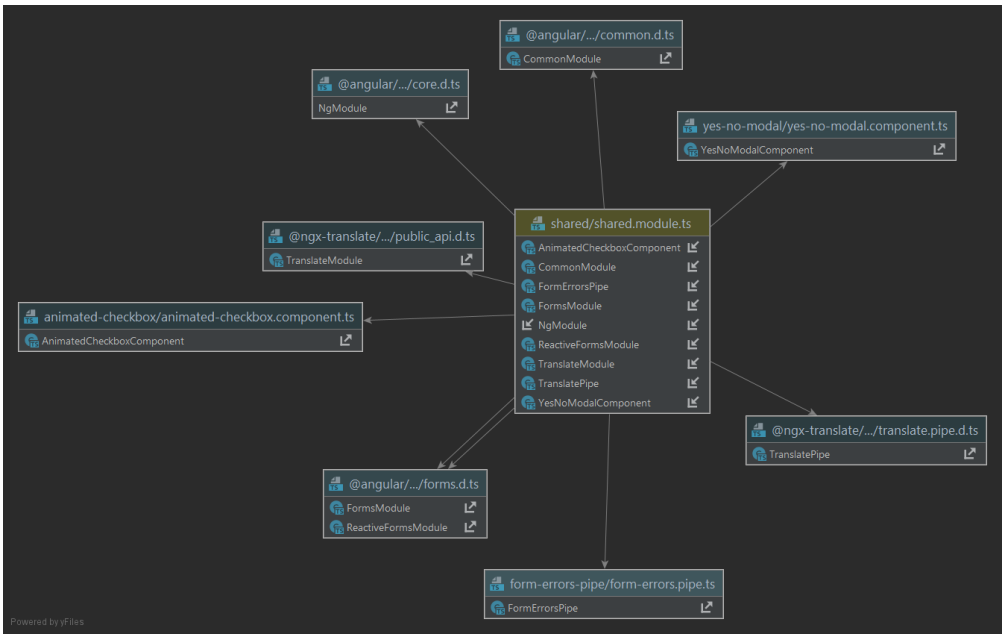


Рисунок 6.3 – Модуль бібліотеки shared-lib

Модуль містить наступні компоненти, що використовуються більшістю мікрофронтендів:

- AnimatedCheckboxComponent – компонент для отримання вводу користувача;
- FormErrorsPipe – функція виводу да перекладу помилок при вводі;
- YesNoModalComponent – модальне вікно підтвердження дії.

6.4 Мікрофронтенд Mfe1: core модуль

Мікрофронтенд mfe1 – це найбільша частина застосунку. Він пердставляє собою портал – понель куди користувач потрапляє після авторизації. Даний мікрофронтенд містить набільше бізнес логіки. Насправді це мікрофронтенд варто було б розділити ще на інші частини, однак для цього знадобилося б ще більше ресурсів. Розглянемо один з основних модулів цього мікроінтерфейсу – модуль core що містить у собі основну частину бізнес логіки (рис. 6.4).

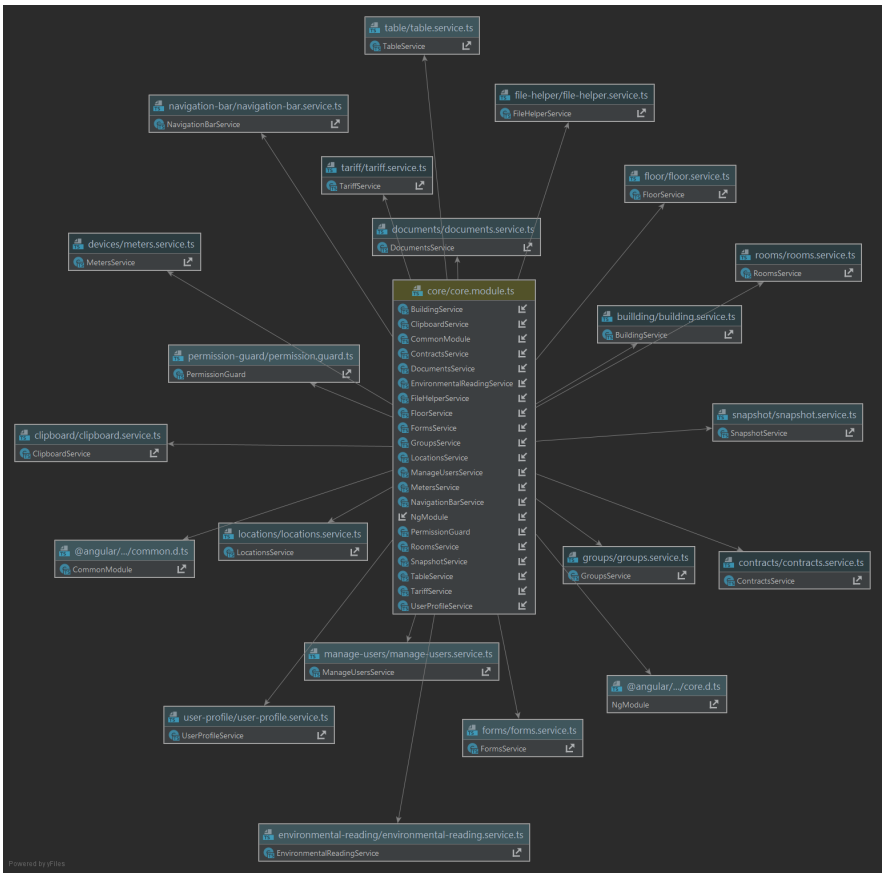


Рисунок 6.4 – Модуль core

Нище, у таблиці 6.1 наведено перелік усії сервісів що входять до цього модуля. Варто зазначити що даний модуль є аналогічним до модуля base-core

застосунку shell. Таким чином за архітектури мікрофрнтендів кожен застосунок має свої власні сервіси.

Таблиця 6.1 – Сервіси модуля shared.

Сервіс	**Призначення**
BuildingService	CRUD будівель
ContractsSercice	CRUD контрактів
ClipboardSercice	Копіювання у буфер користувача
DevicesService	CRUD лічильників
DocumentsService	CRUD документів
EnvironmentalReadingService	CRUD даних кімнати
FloorService	CRUD поверхів
FormsService	Динамічна побудова форм/полів вводу користувача
GroupsService	CRUD користувацьких груп
FileHelperService	Завантаження файлів
LocationsService	CRUD локацій
ManageUsersService	CRUD користувачів
NavigationBarService	Робота панелі навігації
PermissionGuardService	Перевірка прав користувачів та дозвіл виконувати певні діхї у системі
RoomsService	CRUD кімнат
SnapshotService	CRUD показання лічилькиків
TableService	Побудова таблтці
TariffService	CRUD тарифів
UserProfileService	CRUD профіля користувача

Сервіси CRUD, реалізують інтерфейс - CrudInterface. Він забезпечує виконання чотирьох методів: створення, читання, оновлення та видалення. Налаштування цього інтерфейсу забезпечує виконання всіх операцій. Надалі ці сервіси будуть використовуватися ефектами NgRx, і в залежності від запитаної дії будуть викликані відповідні сервісні методи, тому цей елемент дуже важливий. Наприклад, коли викликається процедура оновлення, для оновлення збережених даних викликається метод оновлення обслуговування номерів. Решта процесу працює так само.

Особливої уваги заслуговує NavigationBarService, яка є службою навігації по будівлях. Його принцип роботи полягає в тому, щоб слідувати за поточною URL-адресою та повертати під-адресу навігації, пов’язану з цією URL-адресою. Файл конфігурації містить усі доступні підрозділи, тому нові розділи можна легко додавати. Сервіс безпеки PermissionGuardService був створений для блокування доступу до певних ресурсів, до яких авторизовані користувачі не мають доступу. Ось як це працює: спочатку всі привілеї користувача (усі системні ресурси, до яких користувач має доступ) витягуються та порівнюються з поточними точками шляху. Якщо елемент навігації не входить до дозволів користувача, він буде заблоковано, і користувач не зможе отримати доступ до сторінки. Зауважте, що спроба отримати доступ до ресурсу, який недоступний через термінологію адреси браузера, перенаправляє користувача на доступний ресурс.

6.5. Мікрофронтенд Mfe1: portal-shared модуль

Попри те що застосунок має свою бібліотеку компонентів, не всі компоненти слід виносити туди. Наприклад у системі можуть бути компоненти які необхідні лише в одній частині. У випадку мікрофронтендів – немає потреби виносити спільні компоненти одного мікрофронтенда у глобальну бібліотеку – це навпаки зробить гірше, так як розмір бібліотеки збільшиться, і, як наслідок, це збільшить час на початкове завантаження сайту. Тому усі спільні компоненти mfe1 було вирішено тримати у його власних рамках, а саме в модулі portal-shared (рис. 6.5).

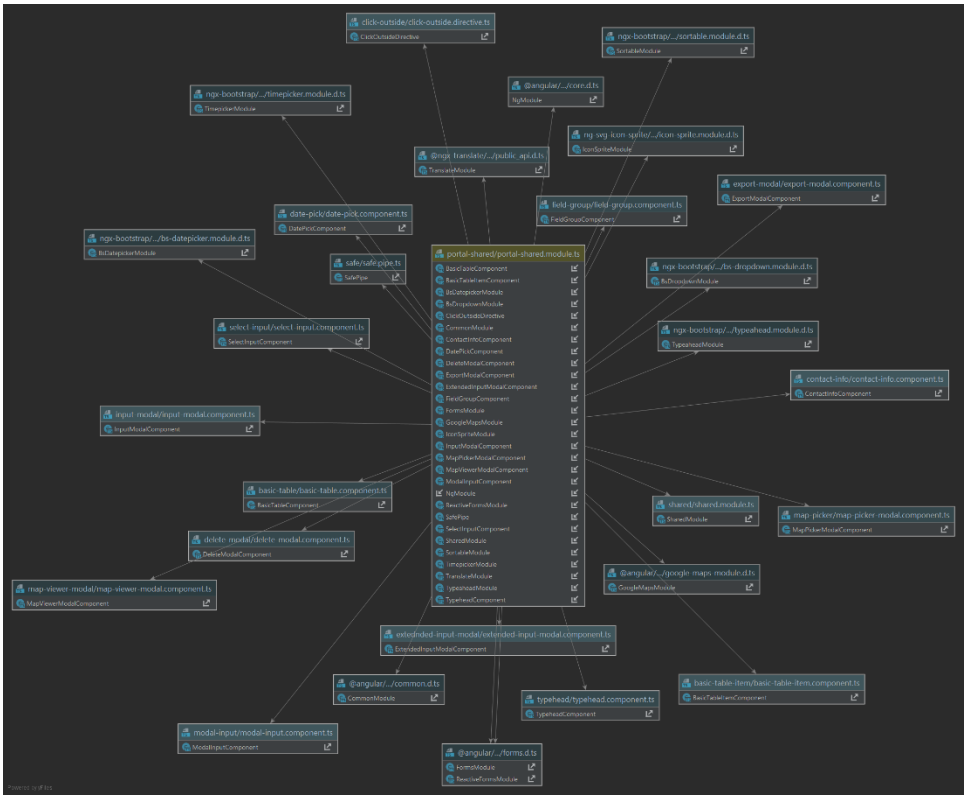


Рисунок 6.5 – Модуль portal-shared

На відміну від попередньо розглянутого модуля модуль містить переважно компоненти для створення користувацького інтерфейсу, такі як

таблиці та модальні вікна, а також елементи що слугують для вводу користувача. Перелік всіх комнпонентів містить таблиця 6.2.

Таблиця 6.2 – Компоненти модуля portal-shared

Компонент	**Призначення**
BasicTable	Основна таблиця
BasicTableItem	Рядок таблиці
ContactInfo	Контактна інформація користувача
DatePick	Ввід дати
DeleteModal	Попередження про видалення сутності
ClickOutsideDirective	Обробка кліку що був не на поточному елементі
ExportModal	Експорт даних
InputModal	CRUD операції
FieldGroup	Групування полів форми
ExtendedInputModад	Модальне вікно з можливістю катомізації
MapPicker	Карта Google
MapViewerModal	Перегляд локації
SelectInput	Ввід користувача з перелічених варіантів
TypeHead	Компонент для вводу користувача з можливістю пошуку та вибору з запропонованих варіантів

Всі перераховані компоненти розроблені відповідно до прикладного проекту. Зауважте, що деякі компоненти служать будівельними блоками для

інших компонентів у цьому пристрої. Таким чином, він не буде використовуватися поза блоком. Одними з найважливіших компонентів, які використовуються в програмі, є компонент таблиці (BasicTable) і компонент модального вікна введення (InputModal). Зазначений компонент зустрічається найчастіше. BasicTable — це таблиця з можливістю автоматичного створення стовпців і рядків. Структура таблиці будується на основі полів об’єктів, переданих до елементів таблиці. Крім того можна вибрати поля, які потрібно виключити. Зверніть увагу, що логіка виклику модального вікна реалізована в основному компоненті таблиці. Тобто використовувати (містити) його. Масив просто інформує батьківський компонент про певну дію користувача (користувач натискає кнопку, щоб створити нову сутність) і надсилає інформацію, що відображається в масиві, передаючи об’єкт конфігурації. Будь-якому полю в таблиці можна призначити тип, наприклад координату. Крім того, компонент таблиці виконує перегортання та сортування. У нижній частині столу управління розташоване реле (рис. 6.7). У кожному полі з назвою стовпця є кнопка для сортування даних за цим полем (рис. 6.6).

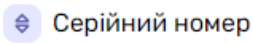


Рисунок 6.6 – Сортування

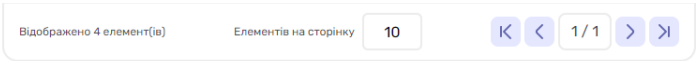


Рисунок 6.7 – Панель пагінації

Крім того, таблиця відповідає за виклик модального вікна, відповідального за редагування, створення та видалення інформації, що відображається в таблиці,

а також надсилання даних, необхідних для обробки запитаної дії (модальне вікно редагування надсилає інформацію про сутність).

Компоненти InputModal і ExtendedInputModal діють як модальні вікна для введення та редагування даних. Ці компоненти дозволяють створювати модальні вікна з будь-якою кількістю полів введення. Щоб компонент працював належним чином, форму сутності (клас), яку потрібно створити або відредагувати, потрібно перемістити до компонента. Крім того, логіка створення для кожного поля пояснюється в ModalInput, який використовується у вищезгаданому компоненті. ModalInput створює поле введення на основі переданого типу. У системі наявні наступні поля вводу:

- Довільний текст;
- Тільки числа;
- Документ;
- Локація;
- Час та дата;
- Вибір у випадаючому списку;
- Поле паролю;
- Пошук та вибір із запропонованих варіантів.

Під ці типи вводу було розроблено відповідні компоненти, наприклад компонент карти дозволяє обрати координати на карті – що значно полегшує знаходження локації. У випадку коли користувачу потрібно знайти та вказати певну сутність у формі що залежить від інших сутностей дуже допомогає TypeHead який дозволяє знайти сутність за текстом який ввів користувач.

6.6. Мікрофронтенд Mfe1: portal модуль

Цей модуль використовує модуль portal-shared як основу для побудови своїх компонентів (рис. 6.8). На відміну від попереднього модуля компоненти цього являються повноцінними сторінками, кожна з яких має свій роут. Даний модуль є точкою входу у мікрофронтенд mfe1 – він позначений у конфігурації webpack як модуль що будується як мікрофронтенд.

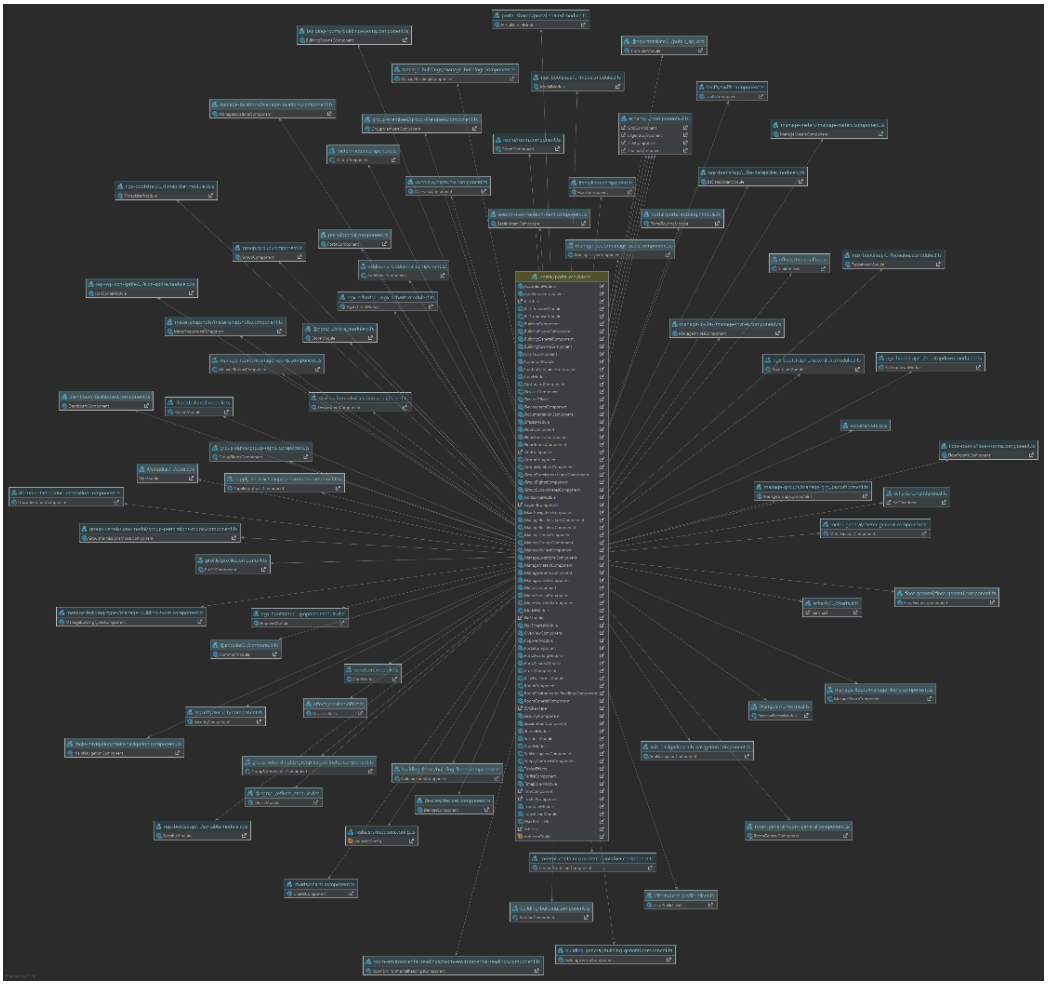


Рисунок 6.8 – Загалька картина модуля portal

Коли користувач проходить авторизацію, яка знаходиться у головному мікрофронтенді shared, починається перехід на роут що веде до мікрофронтенду

mfe1, і, як наслідок відбувається завантаження модуля portal. Таким чином можна сказати що shell є певною мірою обгорткою яка збирає у себе всі мікрофронтенди разом, підвантажуючи їх метаданні та залежності, необхідні для роботи. Усі компоненти що утворюють сторінки відображає таблиця 6.3.

Таблиця 6.3 – Компоненти модуля portal

Компонент	**Сторінка**
Documentation	Документація
MainNavigation	Головна навігація
Overview	Огляд системи
SupplyContract	Договори та документи
Tariffs	Тарифи
SubNavigation	Підрозділи головної навігації
Profile	Профіль користувача
ManageBuildingTypes	Типи будівель
ManageFloors	Поверхи будівлі
ManageGroups	Групи користувачів
ManageInvites	Запрошення користувачів
ManageLocations	Локації
ManageMeters	Лічильники/Iot пристрої
ManageRooms	Кімнати будівлі
ManageUsers	Користувачі системи

Навігація по застосунку відбувається насупним чином: спочатку обгортка переходить на роут що відповідає порталу, потім завантажує його, тобто завантажується відповідний мікрофронтенд.

Висновки до розділу 6

Даний розділ описує поділ на мікрофронетенди системи кабінет “Енерогоменеджера”, а також модулі на які вони поділяються. Отже, було розглянуто наступне:

1. Поділ застосунка на мікрофронтенди.
2. Взаємодія між модулями всередині окремих мікрофронтендів.
3. Реалізацію бізнес логіки через сервіси та модулі
4. Сторінки наявні у системі.
5. Бібліотеку компонентів та спільні компоненти всередині під-програм.

У деяких розділах було акцентовано увагу на значущих елементах системи, таких як таблиці та модальні вікна, бо з цих елементів і складається вся система.

7 ВСТАНОВЛЕННЯ СИСТЕМИ ТА РОБОТА З НЕЮ

7.1. Сервер та інсталяція

Щоб встановити систему програмно-технічне забезпечення вашого сервера має відповідати наступним вимогам:

- Мінімум два ядра процесора та тактова частота 2 ГГц (чим більше цих характеристик, тим краще
- Мінімум 128 мб постійної пам’яті;
- RAM – мінімум 4 Гб;
- Linux або інші системи unix;
- Сервер має бути зі встановленими Git та Docker;

Система розроблялася та тестувалася на наступному програмно-технічному забезмеченні:

- AMD Ryzen 5600x / Apple M1 pro
- RAM 16 gb;
- Windows 11/ MacOs Ventura 13;
- SSD 500 gb

Програма може працювати в двох режимах: запуск програми для розробки та запуск на віддаленому сервері. Різниця між цими режимами полягає в тому, що режим розробки дозволяє автоматично створювати вашу програму, коли виявляються зміни у файлі. Це значно прискорює розробку програми, оскільки програму не потрібно пере-налаштовувати вручну, щоб побачити зміни. Таким чином, сервер, на якому працює веб-додаток, доступний локально, коли програма працює в режимі розробки.

Для того щоб інсталювати систему необхідно під’єднатися до серверу, який буде хостити ПЗ (рис. 7.1).

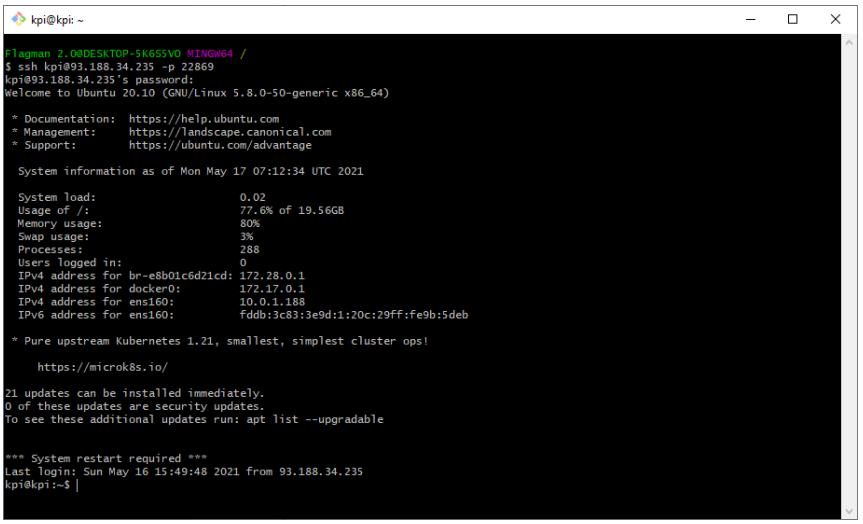


Рисунок 7.1 – Віддалений сервер

Оскільки у корні проекту міститься файл для встановлення самого застосунку, цілком логічним є рішення завантажити застосуноу на віддалений сервер – git clone <repository> (рис. 7.2).

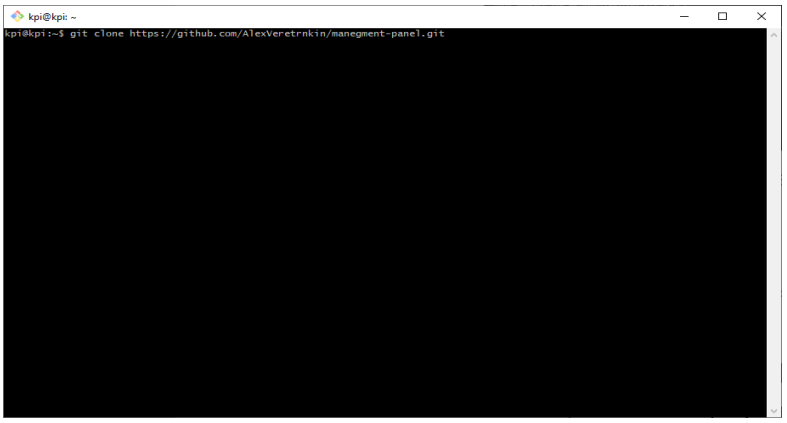


Рисунок 7.2 – Клонування гіт репозиторію

Після завантаження системи на сервер спочатку потрібно отримати доступ до папки проекту та завантажити систему. Для завантаження системи ми

використовуємо Docker який має бути встановленим на сервері. Далі необхідно запустити скрипт sh, який взаємодіє з Docker (рис. 7.3).

Цей скрипт працює так: спочатку нові зміни завантажуються з GitHub, потім створюється образ додатка (докер-образ), потім існуючий контейнер зупиняється та видаляється. На останньому кроці сценарій збірки запускає новий контейнер на основі включеного зображення.

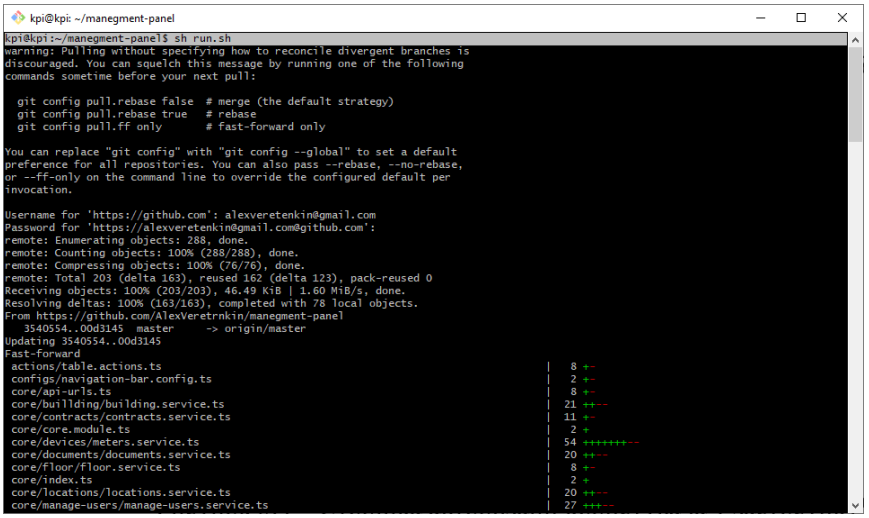


Рисунок 7.3 – Робота скрипта

Після встановлення програми на сервері користувачі отримують доступ до системи. Зазначимо що на прикладі зображена встановлення системи на звичайний сервер – такий процес був продемонстрований щоб емулювати роботу сервісів для розгортання то розробки систем на кшталт Azure Devops чи AWS. Оскільки вся система запускається через докер, переробити скрипт sh не буде проблемою та не займе багато часу.

7.2. Графічне відображення системи

Після авторизації користувач отримує доступ до порталу, тобто до кабінету енергоменеджера. Нові користувачі, які щойно зареєструвалися, також потрапляють у той самий портал (рис 7.4).

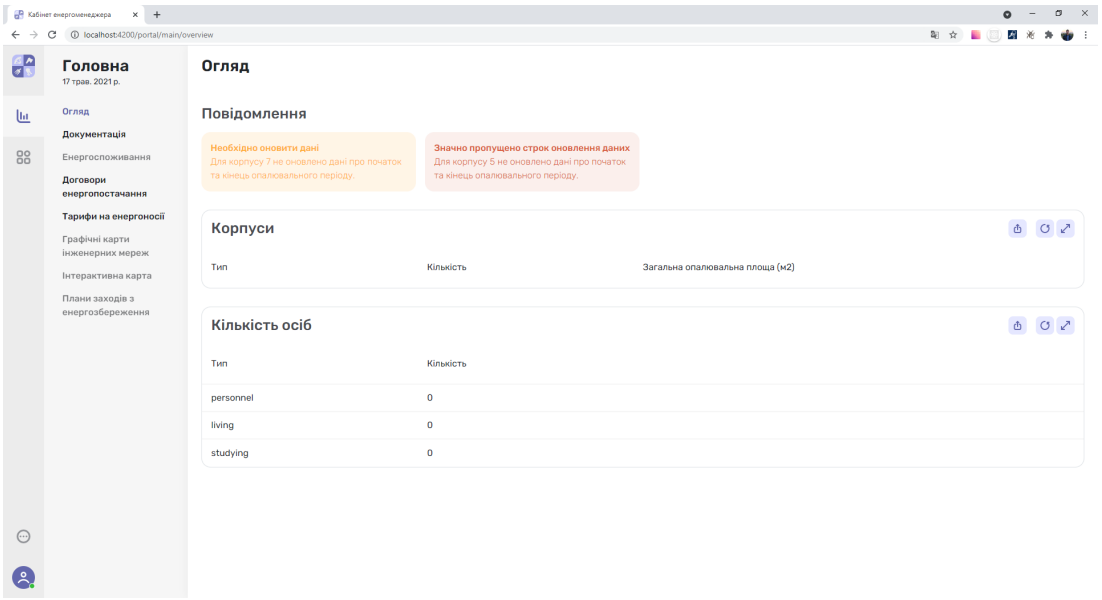


Рисунок 7.4 – Початкова сторінка порталу.

З графічної точки зору систему можна поділити на 3 основні ділянки:

1. Головна навігація – дає змогу вибрати між розділами системи: огляд системи, керування енергоменеджментом та перехід на профіль користувача;
2. Під-меню головної навігації – містить підрозділи головного меню. Певні пункти цього меню згруповані за категоріями.
3. Керування обраною сутністю – перегляд вибраної сторінки, створення, змінення та переглядання інформації.

7.3. Взаємодія користувача з системою

Найпоширенішими елементами системи є таблиці та модальні вікна (рис. 7.5, 7.6). Модальні вікна використовуються для введення, зміни та видалення інформації. З іншого боку, таблиці служать засобом перегляду інформації та містять кнопки для маніпулювання даними, виконання певних дій (введення даних, редагування, завантаження файлів тощо) залежно від контексту.

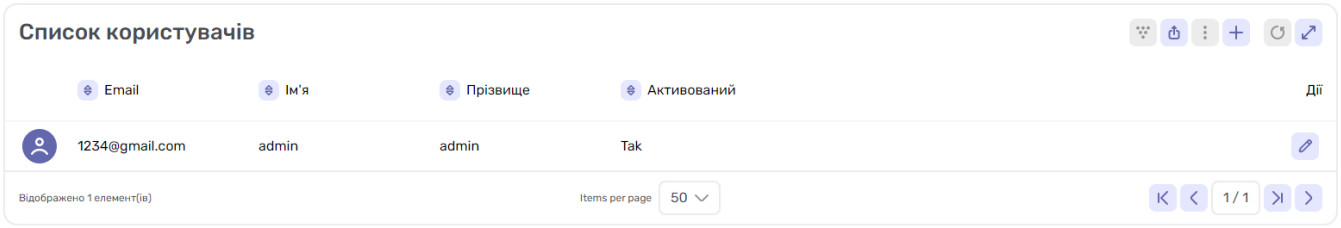


Рисунок 7.5 – Таблиця що відображає список користувачів

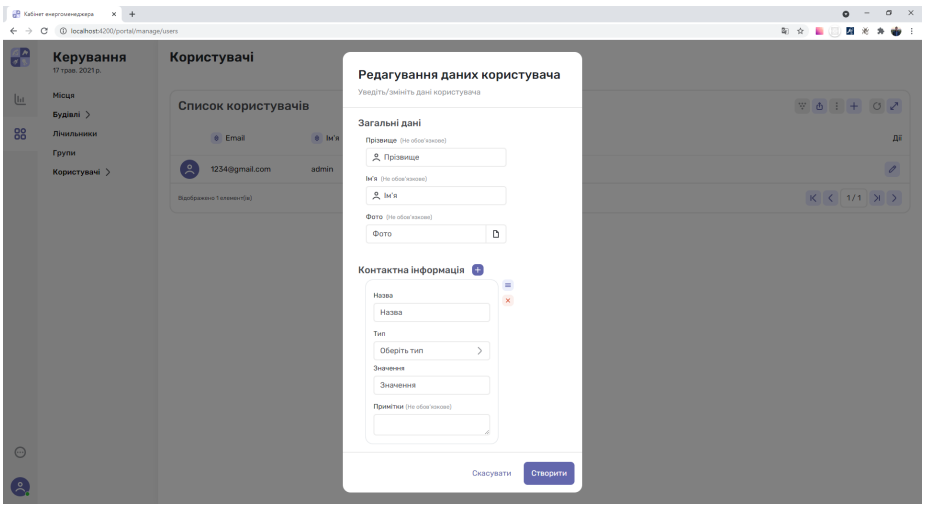


Рисунок 7.6 – Модальне вікно створення/редагування сутності

Повна система наразі включає лише зразок таблиці та наведене вище вікно. Дуже небагато сторінок не мають такої структури. Наприклад, сторінка редагування профілю користувача або сторінка інформації про сутність. Крім

того, одним із найпоширеніших засобів взаємодії з користувачем є типове вікно «Так» або «Ні», яке використовується у разі видалення даних (рис. 7.7). Це робиться для того, щоб запобігти випадковому видаленню даних користувачами.

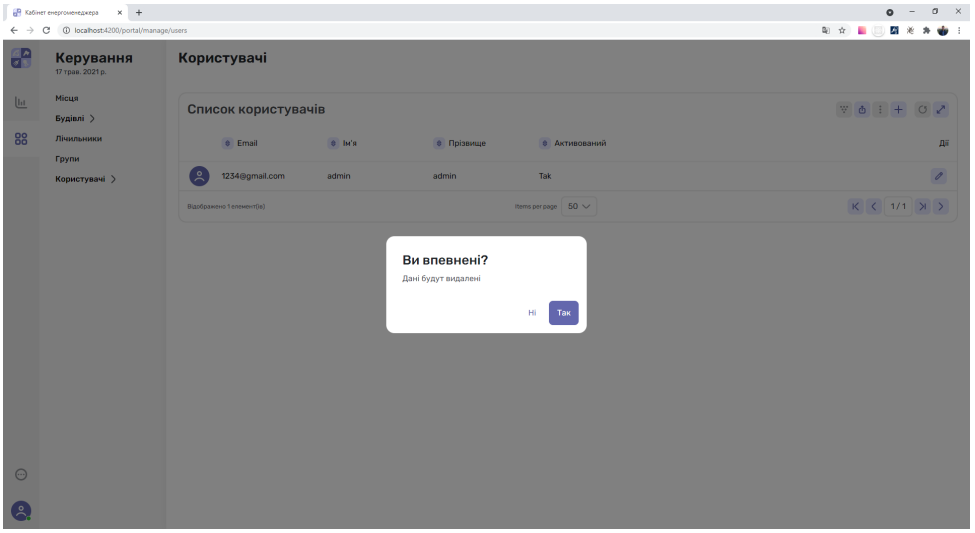


Рисунок 7.7 – Підтвердження дії користувача

Ці способи взаємодії з системою роблять роботу користувача дуже гладкою та легкою для розуміння. Крім того, таблиці та модальні вікна є нескінченно багаторазовими компонентами. Це значно прискорює процес розробки.

Висновки до розділу 7

У даному розділі розглянуто кінцеву систему. Показано способи встановлення системи на віддалений сервер. Вказані рекомендації щодо оптимального технічного забезпечення. Отже в даному розділі було розглянуто наступне:

1. Деплой та оновлення веб додатку на прикладі sh крипта.
2. Технічне забезпечення на якому розробляється система
3. Розглянуті основні елементи системи за допомогою яких здійснюється керування.

8 РОЗРОБКА СТАРТАП ПРОЕКТУ

8.1 Опис ідеї проекту

Даний розділ розглядає можливість розробки проекту як стартапу на основі розробленої системи мікрофронтендів. Головною метою проекту є покращення роботи для енергоменеджерів, а саме: документообіг, моніторинг енергоспоживання, водоспоживання та інших ресурсів, можливість створювати групи користувачів для поділу роботи між ними. Головним завданням стартапу зазвичай є залучення інвесторів та вихід на ринок. Спочатку розглянемо область застосування даного продукту:

- Сучасні житлові комплекси;
- Громадські установи;
- Торгово-розважальні центри.

Ми визначили місця де система може знадобитися, а тому можна визначити цільову аудиторію:

- Насамперед, енергоменеджери;
- Мешканці житлових комплексів;
- Працівники комунальних підприємств.

Продукт має наступну цінність для користувачів:

- Створення груп для користувачів;
- Документообіг в одному місці;
- Автоматичний або ручний збір даних з лічилькиків(на випадок старих будівель, або тих в яких немає розумних лічильників);
- Моніторинг витрат на енергоресурси;
- Побудова певної моделі будинку з його лічильниками.

8.2 Аудит технологій проекту

Проект було побудовано з використанням найновіших технологій що наразі доступні. Повний стек технологій перелічено у таблиці 8.1.

Таблиця 8.1 – Перелік технологій

№	Технологія	Призначення	Наявність	Доступність
1	JavaScript	Мова програмування для написання клієнтських та серверних додатків	Наявна технологія	Доступна
2	TypeScript	Основна мова за допомогою якої написано весь проект. Є значним покращенням мови JavaScript.	Наявна технологія	Доступна
3	Angular	Фреймворк для побудову веб-додатків, підтримує як клієнтський рендеринг, так і серверний	Наявна технологія	Доступна
4	NgRx	Бібліотека що реалізуж управління станом застосунку	Наявна технологія	Доступна
5	Docker	Інструмент для контейнеризації	Наявна технологія	Доступна

Продовження таблиці 8.1

6	Ngx-Bootstrap	Бібліотека що спрощує побудову графічних елементів та містить готові рішення для існуючих проблем	Наявна технологія	Доступна
7	Git	Система що дозволяє зберігати код, ділитися їм з іншими людьми, дозволяє розробляти проект декільком людям одночасно	Наявна технологія	Доступна
8	WebPack Module Federation	Технологія збірки мікрофронтендів в один застосунок	Наявна технологія	Доступна

Оскільки було обрано архітектуру мікрофронтендів, то проект можна з легкістю розширювати без проблем зламати ті частини що вже коректно працюють.

8.3 Ринкові можливості розробленої системи

Аналіз можливостей ринку для продукту вимагає аналізу його потреб, щоб на основі них запропонувати продукт, які покращують ситуацію. Оцінка потреб наведена в таблиці 8.2.

Таблиця 8.2 – Оцінка потреб.

Показник	Оцінка
Необхідність продукту	Існує
Кількість продуктів що можна випустити	Обмежень нема(продукт потребує лише засобів для розгортання та збередення інформації)
Динаміка ринку	Зростає, особливо на тлі використання “зеленої” енергії та впровадження розумних будинків
Стандартизація та специфікація	Продукт не потребує стандартизації та специфікації, окрім випадків коли його необхідно експортувати для сторонніх ресурсів(наприклад як підключення відкео з youtube через iframe)
Наявність великої кількості клієнтів яких зацікавить продукт	Клієнтів вистачає

Дані таблиці 8.2 свідчать про те що даний програмний продукт може стати популярним на ринку через те що не треба шукати клієнтів що будуть зацікавлені в ньому, та через простоту впровадження ПЗ.

Жоден продукт не може бути без проблем, або ж без потенційних проблем з якими можна зіткнутися у майбутньому. Таблиця 8.3 наводить деякі з потенційних проблем що можуть виникнути під час впровадження стартапу.

Таблиця 8.3 – Потенційні проблеми та ризики.

№	Фактор	Потенційна загроза	Рішення проблеми	Вірогідність
1	Конкуренти	Ринок вже має подібні системи, або подібні системи розробляються разом з нашою	Аналіз існуючих рішень, виявлення їх переваг та недоліків; внесення змін в продукт	Висока
2	Збереження даних	Коли система генерує та агрегує велику кількість даних зловмисники можуть спробувати вкрасти ці дані(інформація про користувачів, документи, тощо)	Використання двох факторної авторизації, практика регулярної зміни пароля(система має це вимагати від користувачів)	Середня(на початку стартапу, однак збільшуєтьс я з популярніст ю застосунку)
3	Зникнення потреби у програмно му продукті	ПЗ не буде користуватися попитом на ринку	Дана проблема навряд-чи виникне, оскільки енергоресурси необхідно моніторити	Низька

Продовження таблиці 8.3

4	Залучення інвестицій	Найголовніша проблема стартапів – інвестиції. Може скластися така ситуація що стартап не отримає належного фінансування	Необхідно створити рекламну компанію яка залучить багатьох інвесторів.	Висока
5	Складність використання системи для користувача	Система може виявитись досить обширною і користувач не зможе в ній розібратися	Щоб запобігти такому сценарію необхідно зробити онбоардінг для всієї системи та налаштувати службу підтримки	Середня(залежить від користувачів)
6	Брак кадрів(розробників)	У стартам досить складно залучити нових розробників	Перш за все набирати в команду ідейних людей яким цікавий сам проект.	Висока(на жаль, у стартапів досить великий плин кадрів)

Очевидно що системи такого плану завжди будуть мати хоча б декілька проблем з наведеної вище таблиці. Тому проект необхідно весь час розвивати –

не тільки в технічному плані, а й в плані роботи з кінцевим користувачем. Звісно, усі ризики та проблеми неможливо передбачити, проте ті що можна – треба усувати ще на етапі планування стартапу.

Добре спланований стартап завжди має план додавання нового функціоналу – roadmap. Такий план зазвичай включає в себе покращення проектую. Таблиця 8.4 демонструє можливі покращення проекту.

Таблиця 8.4 – Покращення системи.

№	Покращення	Результат покращення	Складність
1	Оновлення інформації в реальному часі за допомогою веб-сокетів	Система стає набагато інтерактивнішою. Енергоменеджери зможуть значно швидше реагувати на зміни у системі	Середня
2	Більша кількість мікрофронтендів	Дане покращення збільшить ізоляцію між частинами системи, що у свою чергу призведе до більш швидкого розгортання.	Середня

Продовження таблиці 8.4

3	Публікація власної бібліотеки у регістр npm	Можливість пере-використати бібліотеку всім командам одночасно. Оновлення бібліотеки оновить всі компоненти у мікрофронтендах	Середня
4	Побудова детального плану кімнат та поверхів будівлі	Можливість візуалізувати план будівлі по кожному поверху	Дуже висока
5	Представлення статистики у вигляді графіків до кожної таблиці системи	Краща візуалізація інформації	Середня

Як бачимо система може бути покращена у багатьох аспектах. Однак тут необхідно вибрати які покращення найбільш пріоритетні. У випадку стартапу варто запитувати користувачів або інвесторів щодо покращень які мають бути зроблені у першу чергу.

Після аналізу продукту можемо приступити до порівняння переваг та недоліків стартапу. Порівняння наведено у таблиці 8.5

Таблиця 8.5 – Порівняльний аналіз системи

№	Фактор	Оцінка	Рейтинг шляхом порівняння						
			-3	-2	-1	0	+1	+2	+3
1	Результативність	5						+	
2	Можливості	8						+	
3	Сучасність	10							+
4	Тех-підтримка	9			+				
5	Простота для користувача	8					+		
6	План розвитку продукту	10						+	
7	Клієнтоорієнтовність	12					+		

Розроблений продукт має свої переваги над конкурентами або подібними системами. З мінусів можна виділити тех-підтримку, орієнтованість на клієнта та простоту користування.

Висновки до розділу 8

Розділ представляє собою представлення розробленого продукту як стартап проекту. У розділі було описано потенційних клієнтів системи та області де вона може застосовуватись. У табличному форматі надано технічний аудит системи, аналіз потенційних проблем та шляхи їх вирішення. Також було надоно план можливих покращень системи та їх результатів.

Як наслідок проведено порівняльний аналіз системи з іншими та виявлено її переваги та недоліки над іншими системами. Відзначимо що мікрофронтендна архітектура робить проект технологічнішим за конкурентів та надає можливість долучати аутсорс команди для пришвидшення розробки.

ВИСНОВКИ

Результатом виконання магістерської дисертації можна вказати наступні тези:

1. Здійснено аналіз подібних та схожих систем, виявлено їх слабкі та сильні сторони;
2. Розглянуто основні ідеї мікрофронтендів, виявлено їх слабкі та сильні сторони, проаналізовано патерни їх реалізації – сереверний рендеринг та клієнтський;
3. Підібрано набір інструментів для розробки та стек технологій що забезпечать ефективну розробку;
4. На основі аналізу архітектури мікрофронтендів розроблено систему застосунків(мікроінтерфейсів) що збираються в один;
5. Кожний мікрофронтенд збудований таким чином щоб його можна було легко розширити та, якщо це доцільно розділити на інші мікрофронтенди;
6. Розроблено механізм розгортання застосунку який легко адаптувати під більшість платформ;
7. Систему налагоджено та протестовано як на локальній машині, так і на віддаленому сервері;
8. Розглянуто можливість впровадження системи як стартап проект. Проаналізовано можливості системи, попит ринку, потенційні проблеми що виникнуть під час реалізації стартапу та запропоновано шляхи їх вирішення. Сфомовано план покращення системи.

Систему побудовано з використанням рендерингу на стороні клієнта через те що вона має портал, який недоступний до індексації(необхідно бути авторизованим щоб туди потрапити), тому перевага серверного рендеренгу у даній системі не діє.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Geers M. Micro frontends in action. New York : Manning Publications Go, 2020. 296 с.
2. Angular. *Angular*. URL: https://angular.io (дата звернення: 08.11.2022).
3. HTML Tutorial. *W3Schools Online Web Tutorials.* URL: https://www.w3schools.com/html (дата звернення: 13.10.2022).
4. CSS | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/ru/docs/web/css (дата звернення: 01.11.2022).
5. JavaScript | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-us/docs/web/javascript (дата звернення: 03.11.2022).
6. Jooby – smart lighting and remote metering of power, gas, heat, water consumption. *Jooby*. URL: https://jooby.eu/ (дата звернення: 09.10.2022).
7. Help Your Customers Track, Manage, and Save on Energy Usage with Alexa’s New Energy Dashboard (Coming Soon). *alexa-blog*. URL: https://developer.amazon.com/en-US/blogs/alexa/device-makers/2020/09/Help-Your-Customers-Track-Manage-and-Save-on-Energy-Usage-with-Alexas-New-Energy-Dashboard-Coming-Soon (дата звернення: 10.10.2022).
8. CET Inc. *CET Inc*.URL: https://www.cet-global.com/ (дата звернення: 11.10.2022).
9. NgRx Docs. *NgRx Docs*. URL: https://ngrx.io (дата звернення: 10.11.2022).
10. Мартін Р. Чиста архітектура. Харків : Ранок, 2018. 368 с.

ДОДАТОК А

Архітектура та патерни мікрофронтендів, принципи їх реалізації, слабкі та сильні сторони

Специфікація

УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ ТВ-12мп

Аркушів 2

Київ – 2022

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ ТВ-12мп_2	Записка Веретьонкін О.С.docx	Текстова частина дипломної роботи
Компоненти програми		
УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ ТВ-12мп_2-1	nginx.conf	Файл конфігурації сервкру Nginx
УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ ТВ-12мп_2-2	Dockerfile	Файл контейнеризації застосунку
УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ ТВ-12мп_2-3	run.sh	Скрипт запуску додатка
УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ ТВ-12мп_2-4	basic-table.component.ts	Компонент таблиці
УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ ТВ-12мп_2-5	input-modal.component.ts	Компонент модального вікна вводу

УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ ТВ-12мп_2-6	basic-table.component.html	Розмітка таблиці
УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ ТВ-12мп_2-7	input-modal.component.html	Розмітка модалки

ДОДАТОК Б

Архітектура та патерни мікрофронтендів, принципи їх реалізації, слабкі та сильні сторони

Лістинг програми

УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ ТВ-12мп

Аркушів 18

Київ – 2022

nginx.conf

```
events{}
	http {
	include /etc/nginx/mime.types;
		server {
	listen 8000;
		server_name http://93.188.34.235/;
		root /usr/share/nginx/html;
	index index.html;
		location / {
	try_files $uri $uri/ /index.html;
}
}
}
```

Dockerfile

```
### STAGE 1: Build ###
FROM node:12.7-alpine AS build
WORKDIR /usr/src/app
COPY package.json package-lock.json ./
RUN npm install
COPY . .
RUN npm run build
RUN ls

### STAGE 2: Run ###
FROM nginx:1.17.1-alpine
COPY nginx.conf /etc/nginx/nginx.conf
COPY --from=build /usr/src/app/dist/management-panel /usr/share/nginx/html
```

run.sh

```
#!/bin/bash

git pull

docker build -t angular-docker .

docker stop angular-container
docker rm angular-container

docker run --name angular-container -d -p 8000:8000 angular-docker
```

basic-table.component.ts

```
import {
  AfterViewInit,
```

```
  ChangeDetectionStrategy,
  Component,
  ElementRef,
  EventEmitter,
  HostBinding,
  Input,
  OnChanges,
  Output
} from '@angular/core';
import { animate, state, style, transition, trigger } from '@angular/animations';

import {
  AnimationParamsModel, PaginationTableActionEnum, PermissionsEnum, QueryModel, StoreModel,
  TableDataFieldTypeEnum,
  TableDataModel,
  TableEventsEnum,
  TableItemActionEnum,
  TableOptionsModel, UserProfileModel
} from '@models';
import { ClipboardService, TableService } from '@core';
import { KeyValue } from '@angular/common';
import { ActivatedRoute } from '@angular/router';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';
import { debounceTime, distinctUntilChanged, filter, map, share, throttleTime } from 'rxjs/operators';
import { basePermissionsConfig } from '@configs';
import { FormControl } from '@angular/forms';
import { UntilDestroy, untilDestroyed } from '@ngneat/until-destroy';

@Component({
  selector: 'app-basic-table',
  templateUrl: './basic-table.component.html',
  styleUrls: ['./basic-table.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush,
  animations: [
    trigger('openClose', [
      state('open', style({
        top: '0px',
        left: '0px',
        width: '100vw',
        height: '100vh',
        position: 'absolute',
        padding: '4.5rem',
        background: 'rgba(0, 0, 0, .5)'
      })),
      state('closed', style({
          height: '*',
          width: '*',
          top: '{{ top }}px',
          left: '{{ left }}px',
          padding: '0',
```

```
        background: 'unset'
      }),
      {params: {top: 'unset', left: 'unset'}}
    ),
    transition('open => closed', [
      style({
        background: 'unset'
      }),
      animate('.3s cubic-bezier(0.390,  0.575, 0.565, 1.000)', style({
        position: 'absolute',
        top: '{{ top }}px',
        left: '{{ left }}px',
        height: '*',
        width: '*',
        padding: '0',
      }))
    ]),
    transition('closed => open', [
      style({
        position: 'absolute',
      }),
      animate('.3s cubic-bezier(0.470,  0.000, 0.745, 0.715)', style({
        top: '0px',
        left: '0px',
        width: '100vw',
        height: '100vh',
        position: 'absolute',
        padding: '4.5rem'
      })),
      animate('.1s', style({
        background: 'rgba(0, 0, 0, .5)'
      }))
    ]),
  ]),
 ],
})
@UntilDestroy()
export class BasicTableComponent<T> implements AfterViewInit, OnChanges {
 @Input() data!: TableDataModel<T>;

 @Input() tableOptions!: TableOptionsModel<T>;

 @Input() set paginationQuery(query: QueryModel<T>) {
  if (query) {
   this.query = {
    ...query
   } as QueryModel<T>;
  }
 }

 @Output() pagination: EventEmitter<QueryModel<T>> = new EventEmitter<QueryModel<T>>();
```

```
@Output() tableAction: EventEmitter<TableEventsEnum> = new EventEmitter<TableEventsEnum>();

@Output() itemAction: EventEmitter<KeyValue<TableDataFieldTypeEnum, string>> =
  new EventEmitter<KeyValue<TableDataFieldTypeEnum, string>>();

@Output() public controlAction: EventEmitter<KeyValue<TableItemActionEnum, T>> =
  new EventEmitter<KeyValue<TableItemActionEnum, T>>();

public dataFields!: Map<keyof T, TableDataFieldTypeEnum>;

public totalPages!: number;

public readonly actionEnum: typeof TableEventsEnum = TableEventsEnum;

public readonly fieldTypeEnum: typeof TableDataFieldTypeEnum = TableDataFieldTypeEnum;

public readonly paginationActionEnum: typeof PaginationTableActionEnum = PaginationTableActionEnum;

public isTableModalShown = false;

private query: QueryModel<T> = new QueryModel<T>();

public pageSizeControl: FormControl = new FormControl(this.query.pageSize);

private offsetTop!: number;
private offsetLeft!: number;

constructor(
  private elementRef: ElementRef,
  private tableService: TableService<T>,
  private clipboardService: ClipboardService,
  private activatedRoute: ActivatedRoute,
  private store: Store<StoreModel>
) {
}

@HostBinding('@openClose') get expand(): AnimationParamsModel {
  return this.isTableModalShown ?
    ({
      value: 'open'
    }) :
    ({
      value: 'closed', params: {
        top: this.offsetTop,
        left: this.offsetLeft
      }
    });
}

public ngAfterViewInit(): void {
```

```
  this.initTablePosition();

  this.pageSizeControl.valueChanges
    .pipe(
      distinctUntilChanged(),
      debounceTime(200),
      untilDestroyed(this)
    )
    .subscribe((val: number) => {
      if (val > 0) {
        this.query.pageSize = val;

        this.pagination.emit(this.query);
      }
    });
}

public ngOnChanges(): void {
  if (!this.tableOptions.excludedControlFields) {
    this.tableOptions.excludedControlFields = [TableEventsEnum.edit];
  }

  if (this.data?.tableData && this.data?.tableData.totalSize > 0) {
    this.tableOptions.dataFields = this.tableService.getDataFieldMap(this.data?.tableData.items[0], this.tableOptions);

    this.totalPages = this.calculateTotalPages();
  }
}

public get isActionDisabled(): Observable<boolean> {
  return this.store.select('userProfile')
    .pipe(
      filter(user => !!user?.id),
      map((user: UserProfileModel) => {
        const path: string = this.activatedRoute.snapshot.url[0].path;

        if (basePermissionsConfig.includes(path)) {
          return false;
        }

        const permissions: PermissionsEnum[] = user.permissions?.filter(
          item => item.toLowerCase().includes(path.replace('-', ''))
        )!;

        return !permissions.some(item => item.toLowerCase().includes('edit'));
      }),
      share()
    );
}

public getFieldType(field: keyof T): TableDataFieldTypeEnum {
```

```
  return this.tableOptions.dataFields?.get(field)!;
}

public getHeadingField(field: keyof T): string {
  return 'tableHeadings.' + field.toString();
}

public emitAction(action: TableEventsEnum): void {
  if (action === TableEventsEnum.expand) {
    this.isTableModalShown = !this.isTableModalShown;
  }

  this.tableAction.emit(action);
}

public onCopy(text: string): void {
  this.clipboardService.copy(text);
}

public onItemAction(event: KeyValue<TableDataFieldTypeEnum, string>): void {
  this.itemAction.emit(event);
}

public isActionExcluded(field: TableEventsEnum): boolean {
  return !!this.tableOptions?.excludedControlFields?.includes(field);
}

public paginationAction(action: PaginationTableActionEnum): void {
  switch (action) {
    case PaginationTableActionEnum.next:
      if (this.query.pageNumber !== this.totalPages) {
        this.query.pageNumber += 1;

        this.pagination.emit(this.query);
      }
      break;
    case PaginationTableActionEnum.previous:
      if (this.query.pageNumber !== 1) {
        this.query.pageNumber -= 1;

        this.pagination.emit(this.query);
      }
      break;
    case PaginationTableActionEnum.last:
      this.query.pageNumber = this.totalPages;

      this.pagination.emit(this.query);
      break;
    case PaginationTableActionEnum.first:
      this.query.pageNumber = 1;
```

```
      this.pagination.emit(this.query);
      break;
    }
  }

  public tackByItems(item: any): any {
    return item?.id;
  }

  private calculateTotalPages(): number {
    return Math.ceil(this.data.tableData.totalSize / this.query.pageSize);
  }

  private initTablePosition(): void {
    this.offsetTop = this.elementRef.nativeElement.offsetTop;
    this.offsetLeft = this.elementRef.nativeElement.offsetLeft;
  }

  public onControlAction(event: KeyValue<TableItemActionEnum, T>, item: T): void {
    this.controlAction.emit(event);
  }

  public toggleSortByFilter(field: keyof T): void {
    if (this.isSortedDown(field)) {
      const newOrder: string[] = [...this.query.orderBy!] as string[];
      newOrder[this.query.orderBy?.indexOf(field)!] = `-${field}`;

      this.query.orderBy = newOrder;
    } else if (this.isSortedUp(field)) {
      this.query.orderBy = this.query.orderBy?.filter(item => item !== `-${field}`);
    } else if (this.isNotSorted(field)) {
      this.query.orderBy = this.query?.orderBy ? [...this.query?.orderBy, field] : [field];
    }

    this.pagination.emit(this.query);
  }

  public isSortedUp(field: keyof T): boolean {
    return !!this.query.orderBy?.includes(`-${field}`);
  }

  public isSortedDown(field: keyof T): boolean {
    return !!this.query.orderBy?.includes(field);
  }

  public isNotSorted(field: keyof T): boolean {
    return !this.query.orderBy?.includes(field);
  }
}
```

input-modal.component.ts

```
import { ChangeDetectionStrategy, Component, Input, OnInit } from '@angular/core';
import { CoordinatesModel, FieldInputModel, InputModalModel, ModalActionEnum, ModalActionModel } from '@models';
import { FormControl, FormGroup } from '@angular/forms';
import { BsModalRef, BsModalService } from 'ngx-bootstrap/modal';
import { FormsService } from '@core';
import { KeyValue } from '@angular/common';
import { MapPickerModalComponent } from '../map-picker/map-picker-modal.component';
import { take } from 'rxjs/operators';

@Component({
  selector: 'app-input-modal',
  templateUrl: './input-modal.component.html',
  styleUrls: ['./input-modal.component.scss'],
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class InputModalComponent<T> implements OnInit {
  @Input() public inputData!: InputModalModel<T>;

  public modalFormGroup!: FormGroup;

  public fieldInputs: FieldInputModel<T>[] = [];

  public readonly actionTypes: typeof ModalActionEnum = ModalActionEnum;

  private filesMap: Map<string, File> = new Map<string, File>();

  private selectionsMap: Map<string, any> = new Map<string, any>();

  public get isDeleteButtonShown(): boolean {
    return Object.values(this.inputData.inputModel).some(v => !!v);
  }

  constructor(
    private modalRef: BsModalRef,
    private formsService: FormsService<T>,
    private modalService: BsModalService
  ) { }

  public ngOnInit(): void {
    this.modalFormGroup = this.formsService.getFormGroupFromModel(
      this.inputData.inputModel,
      this.inputData.excludedFields as string[],
      this.inputData.fieldTypes
    );

    this.fieldInputs = this.formsService.getInputFields(
      this.inputData.fieldTypes!,
      this.inputData.inputModel,
      this.inputData.excludedFields as string[]
    );
```

```
}

public getFormControl(fromControlName: keyof T): FormControl {
  return this.modalFormGroup.get(fromControlName as string) as FormControl;
}

public close(): void {
  this.modalRef.hide();
}

public action(modalAction: ModalActionEnum = ModalActionEnum.create): void {
  const files: {[key: string]: File} = {};
  const selections: {[key: string]: any} = {};

  if (this.inputData.inputModel[this.fieldInputs[0].formControlName] && modalAction !== ModalActionEnum.delete) {
    modalAction = ModalActionEnum.edit;
  }

  [...this.filesMap.keys()].forEach((key: string) => {
    files[key] = this.filesMap.get(key)!;
  });

  [...this.selectionsMap.keys()].forEach((key: string) => {
    selections[key] = this.selectionsMap.get(key)!;
  });

  const action: ModalActionModel<T> = {
    action: modalAction,
    data: {
      ...this.inputData.inputModel,
      ...this.modalFormGroup.getRawValue(),
      ...this.getCoordinates(),
      ...files,
      ...selections
    }
  };

  this.modalRef.onHidden.emit(action);

  this.close();
}

public onFileChange(fileMap: KeyValue<string, File>): void {
  this.filesMap.set(fileMap.key, fileMap.value);
}

public onCoordinatesChange(): void {
  this.modalService.show(MapPickerModalComponent, {
    initialState: {
      coordinates: this.getCoordinates()
    }
```

```
    })
      .onHidden
      .pipe(take(1))
      .subscribe((data: CoordinatesModel) => {
        if (data.coordinates) {
          this.modalFormGroup.get('coordinates')?.setValue(data.coordinates);
        }
      });
  }

  private getCoordinates(): CoordinatesModel {
    const coordinates: string = this.modalFormGroup.get('coordinates')?.value;

    if (coordinates) {
      const [latitude, longitude]: string[] = coordinates.split(' ');

      return {
        coordinates: coordinates!,
        latitude: parseFloat(latitude),
        longitude: parseFloat(longitude)
      };
    }

    return {} as CoordinatesModel;
  }

  public onTypeheadSelection(event: KeyValue<string, any>): void {
    this.selectionsMap.set(event.key, event.value);
  }
}
```

input-modal.component.html

```
<section class="modal__header pb-19 pt-32 pl-32 pr-44">
  <h4 class="mb-8">{{inputData.heading}}</h4>
  <p class="text-regular text_grayscale-neutral m-0">{{inputData.subheading}}</p>
</section>

<form class="pt-8 pb-16 pr-32 pl-32" [formGroup]="modalFormGroup">
  <app-modal-input
    class="mt-8 d-block modal__input"
    *ngFor="let input of fieldInputs"
    [inputData]="input"
    [translation]="inputData.translation ? inputData.translation + '.' : ''"
    [control]="getFormControl(input.formControlName)"
    (fileChange)="onFileChange($event)"
    (coordinatesChange)="onCoordinatesChange()"
    (typeheadSelection)="onTypeheadSelection($event)"
  >
  </app-modal-input>
</form>
```

```
<section class="modal__footer pt-19 pb-16 pr-16 pl-16 d-flex justify-content-end align-content-center">
  <button class="button button_error-neutral modal__delete-button border-radius_8 h-48 pt-12 pb-12 pl-16 pr-16 mr-auto
              d-flex align-items-center"
          *ngIf="isDeleteButtonShown"
          (click)="action(actionTypes.delete)"
  >
    <svg-icon-sprite
      class="mr-8"
      [src]="'assets/sprites/sprite.svg#bin-icon'"
      [classes]="'icon icon_16 icon_grayscale-white'"
    >
    </svg-icon-sprite>

    Видалити
  </button>

  <button class="button button-interaction_outlined button_transparent h-48 pt-12 pb-12 pl-16 pr-16"
          (click)="close()"
  >
    Скасувати
  </button>

  <button class="button button-interaction border-radius_8 h-48 pt-12 pb-12 pl-16 pr-16 ml-10"
          [disabled]="modalFormGroup.invalid"
          (click)="action()"
  >
    {{ isDeleteButtonShown ? 'Зберегти' : (inputData.actionButtonText || 'Створити')}}
  </button>
</section>
```

basic-table.component.html

```
<section class="table border-radius_16 d-flex flex-column flex-1"
         [class.table-footer__filter]="!isTableModalShown"
>
  <div class="h-64 pl-16 pr-16 d-flex align-items-center">
    <h4 class="heading mb-0 mr-auto">{{data?.heading}}</h4>

    <div class="d-flex align-items-center">
      <div class="disabled-item bg-lightest border-radius_8 cursor-pointer ml-4 h-32 w-32
                d-flex align-items-center justify-content-center"
           *ngIf="!isActionExcluded(actionEnum.filter)"
           (click)="emitAction(actionEnum.filter)"
      >
        <svg-icon-sprite
          [src]="'assets/sprites/sprite.svg#dots-triangle-icon'"
          [classes]="'icon icon_16 icon_primary-normal'"
        >
        </svg-icon-sprite>
```

```
</div>

<div class="bg-lightest border-radius_8 cursor-pointer ml-4 h-32 w-32
    d-flex align-items-center justify-content-center"
    [class.disabled-item]="isActionDisabled | async"
    *ngIf="!isActionExcluded(actionEnum.export)"
    (click)="emitAction(actionEnum.export)"
>
  <svg-icon-sprite
    [src]="'assets/sprites/sprite.svg#export-icon'"
    [classes]="'icon icon_16 icon_primary-normal'"
  >
  </svg-icon-sprite>
</div>

<div class="disabled-item bg-lightest border-radius_8 cursor-pointer ml-4 h-32 w-32
          d-flex align-items-center justify-content-center"
    *ngIf="!isActionExcluded(actionEnum.more)"
    (click)="emitAction(actionEnum.more)"
>
  <svg-icon-sprite
    [src]="'assets/sprites/sprite.svg#more-icon'"
    [classes]="'icon icon_16 icon_primary-normal'"
  >
  </svg-icon-sprite>
</div>

<div class="bg-lightest border-radius_8 cursor-pointer ml-4 h-32 w-32 button-interaction
    d-flex align-items-center justify-content-center"
    [class.disabled-item]="isActionDisabled | async"
    *ngIf="!isActionExcluded(actionEnum.add)"
    (click)="emitAction(actionEnum.add)"
>
  <svg-icon-sprite
    [src]="'assets/sprites/sprite.svg#plus-icon'"
    [classes]="'icon icon_16 icon_primary-normal'"
  >
  </svg-icon-sprite>
</div>

<div class="bg-lightest border-radius_8 cursor-pointer ml-4 h-32 w-32 button-interaction
    d-flex align-items-center justify-content-center"
    *ngIf="!isActionExcluded(actionEnum.edit)"
    (click)="emitAction(actionEnum.edit)"
>
  <svg-icon-sprite
    [src]="'assets/sprites/sprite.svg#edit-icon'"
    [classes]="'icon icon_16 icon_primary-normal'"
  >
  </svg-icon-sprite>
</div>
```

```
    <div class="disabled-item bg-lightest border-radius_8 cursor-pointer ml-16 h-32 w-32
          d-flex align-items-center justify-content-center"
      *ngIf="!isActionExcluded(actionEnum.refresh)"
      (click)="emitAction(actionEnum.refresh)"
    >
      <svg-icon-sprite
      [src]="'assets/sprites/sprite.svg#refresh'"
      [classes]="'icon icon_16 icon_primary-normal'"
      >
      </svg-icon-sprite>
    </div>

    <div class="bg-lightest border-radius_8 cursor-pointer ml-4 h-32 w-32 button-interaction
      d-flex align-items-center justify-content-center"
      *ngIf="!isActionExcluded(actionEnum.expand)"
      (click)="emitAction(actionEnum.expand)"
    >
      <svg-icon-sprite
      [src]="isTableModalShown ? 'assets/sprites/sprite.svg#shrink-icon' : 'assets/sprites/sprite.svg#expand-icon'"
      [classes]="'icon icon_16 icon_primary-normal'"
      >
      </svg-icon-sprite>
    </div>
  </div>
</div>

<section class="scrollable-container scrollable-container_grayscale-dark
        align-items-start flex-1 d-flex align-content-stretch position-relative"
>
  <div class="d-flex flex-column flex-1">
    <div class="filter-container min-h-64 p-16 d-flex align-items-center flex-wrap"
      *ngIf="data?.activeFilters"
    >
      <div class="filter border-radius_16 mr-8 mb-8 pt-4 pb-4 pl-8 pr-8 d-flex align-items-center">
        <div class="filter__circle rounded-circle h-16 w-16 mr-8"></div>
        <span class="mr-8">Тип авторизації: Пароль</span>
        <svg-icon-sprite
          class="cursor-pointer"
          [src]="'assets/sprites/sprite.svg#cross-circle-icon'"
          [classes]="'icon icon_16 icon_grayscale-darkest'"
        >
        </svg-icon-sprite>
      </div>

      <div class="filter border-radius_16 mr-8 mb-8 pt-4 pb-4 pl-8 pr-8 d-flex align-items-center">
        <div class="filter__circle rounded-circle h-16 w-16 mr-8"></div>
        <span class="mr-8">Розпочата: Від 01.01.2010, До 16.06.2022</span>
        <svg-icon-sprite
          class="cursor-pointer"
          [src]="'assets/sprites/sprite.svg#cross-circle-icon'"
```

```
    [classes]="'icon icon_16 icon_grayscale-darkest'"
   >
   </svg-icon-sprite>
  </div>

  <div class="filter border-radius_16 mr-8 mb-8 pt-4 pb-4 pl-8 pr-8 d-flex align-items-center">
   <div class="filter__circle rounded-circle h-16 w-16 mr-8"></div>
   <span class="mr-8">Тип авторизації: Пароль</span>
   <svg-icon-sprite
    class="cursor-pointer"
    [src]="'assets/sprites/sprite.svg#cross-circle-icon'"
    [classes]="'icon icon_16 icon_grayscale-darkest'"
   >
   </svg-icon-sprite>
  </div>

  <div class="filter border-radius_16 mr-8 mb-8 pt-4 pb-4 pl-8 pr-8 d-flex align-items-center">
   <div class="filter__circle rounded-circle h-16 w-16 mr-8"></div>
   <span class="mr-8">Розпочата: Від 01.01.2010, До 16.06.2022</span>
   <svg-icon-sprite
    class="cursor-pointer"
    [src]="'assets/sprites/sprite.svg#cross-circle-icon'"
    [classes]="'icon icon_16 icon_grayscale-darkest'"
   >
   </svg-icon-sprite>
  </div>
 </div>

 <div class="head divider-border position-sticky divider-border_with-last color-black h-72 d-flex align-items-center"
    [class.pl-20]="!tableOptions.isCheckboxShown"
 >
  <div class="w-64 h-100 flex-shrink-0 align-items-center justify-content-center"
     [class.d-none]="!tableOptions.isCheckboxShown"
     [class.d-flex]="tableOptions.isCheckboxShown"
  >
   <app-animated-checkbox></app-animated-checkbox>
  </div>

  <div class="w-64 h-100 d-flex flex-shrink-0 align-items-center justify-content-center"
     *ngIf="tableOptions?.expandable"
  ></div>

  <div class="table-item d-flex flex-shrink-0 align-items-center order-1"
     *ngFor="let field of tableOptions?.dataFields?.keys()"
     [class.w-64]="getFieldType(field) === fieldTypeEnum.photo"
     [ngStyle]="{
       'order': getFieldType(field) === fieldTypeEnum.photo ? 0 : 1
     }"
  >
   <ng-container *ngIf="getFieldType(field) !== fieldTypeEnum.photo">
    <div class="method-card__action bg-lightest icon_24 mr-7
```

```
                d-flex flex-shrink-0 align-items-center justify-content-center"
          [class.bg-primary-light]="isSortedUp(field) || isSortedDown(field)"
          (click)="toggleSortByFilter(field)"
        >
          <svg-icon-sprite
            class="cursor-pointer"
            *ngIf="isNotSorted(field) && !isSortedUp(field) && !isSortedDown(field)"
            [src]="'assets/sprites/sprite.svg#filter-toggle-icon'"
            [classes]="'icon icon_12 icon_primary-normal'"
          >
          </svg-icon-sprite>

          <svg-icon-sprite
            class="cursor-pointer"
            *ngIf="isSortedUp(field)"
            [src]="'assets/sprites/sprite.svg#triangle-up-icon'"
            [classes]="'icon icon_12 icon_grayscale-white'"
          >
          </svg-icon-sprite>

          <svg-icon-sprite
            class="cursor-pointer rotate-180deg"
            *ngIf="isSortedDown(field)"
            [src]="'assets/sprites/sprite.svg#triangle-up-icon'"
            [classes]="'icon icon_12 icon_grayscale-white'"
          >
          </svg-icon-sprite>
        </div>

        <span class="text-regular" [textContent]="getHeadingField(field) | translate"></span>
      </ng-container>
    </div>

    <div class="text-right ml-auto mr-16 order-last"
        *ngIf="tableOptions?.itemActionsIcons"
    >
      Дії
    </div>
  </div>

  <div class="text_grayscale-dark p-16 w-100 d-flex justify-content-center align-items-center"
      *ngIf="data.tableData && data.tableData?.totalSize === 0"
  >
    Немає даних
  </div>

  <app-basic-table-item
    *ngFor="let item of data?.tableData?.items; let i = index; trackBy: tackByItems"
    [data]="item"
    [index]="i"
    [itemOptions]="tableOptions"
```

```
    [isControlsDisabled]="(isActionDisabled | async)!"
    (itemCopy)="onCopy($event)"
    (itemAction)="onItemAction($event)"
    (controlAction)="onControlAction($event, item)"
  ></app-basic-table-item>
 </div>
</section>

<div class="table-footer heading mt-auto h-62 pl-16 pr-16 w-100 d-flex align-items-center justify-content-between">
 <span class="label-regular">Відображено {{data?.tableData?.pageSize}} елемент(ів)</span>

 <div class="d-flex align-items-center">
  <span class="label-regular mr-8">Елементів на сторінку</span>

  <input class="base-input base-input_grayscale-light w-64 d-flex text-center"
       type="number"
       min="1"
       [formControl]="pageSizeControl">
 </div>

 <div class="h-32 d-flex align-items-center">
  <div class="bg-lightest border-radius_8 h-32 w-32 mr-4 d-flex justify-content-center align-items-center"
     (click)="paginationAction(paginationActionEnum.first)"
  >
   <svg-icon-sprite
    class="cursor-pointer"
    [src]="'assets/sprites/sprite.svg#next-skip'"
    [classes]="'icon icon_16 icon_primary-normal'"
   >
   </svg-icon-sprite>
  </div>

  <div class="bg-lightest border-radius_8 h-32 w-32 mr-4 d-flex justify-content-center align-items-center"
     (click)="paginationAction(paginationActionEnum.previous)"
  >
   <svg-icon-sprite
    class="cursor-pointer rotate-90deg"
    [src]="'assets/sprites/sprite.svg#arrow-icon'"
    [classes]="'icon icon_16 icon_primary-normal'"
   >
   </svg-icon-sprite>
  </div>

  <div class="table-footer__filter border-radius_8 h-40 mr-4 pt-6 pb-6 pl-12 pr-12">
   <span class="heading">{{data?.tableData?.pageNumber}} / {{totalPages}}</span>
  </div>

  <div class="bg-lightest border-radius_8 h-32 w-32 mr-4 d-flex justify-content-center align-items-center"
     (click)="paginationAction(paginationActionEnum.next)"
  >
   <svg-icon-sprite
```

```
          class="cursor-pointer rotate-90deg-reverse"
          [src]="'assets/sprites/sprite.svg#arrow-icon'"
          [classes]="'icon icon_16 icon_primary-normal'"
        >
        </svg-icon-sprite>
      </div>

      <div class="bg-lightest border-radius_8 h-32 w-32 d-flex justify-content-center align-items-center"
          (click)="paginationAction(paginationActionEnum.last)"
      >
        <svg-icon-sprite
          class="cursor-pointer rotate-180deg"
          [src]="'assets/sprites/sprite.svg#next-skip'"
          [classes]="'icon icon_16 icon_primary-normal'"
        >
        </svg-icon-sprite>
      </div>
    </div>
  </div>
</section>
```

ДОДАТОК В

Архітектура та патерни мікрофронтендів, принципи їх реалізації, слабкі та сильні сторони

Презентація

УКР.НТУУ«КПІ»_НН ІАТЕ_ІПЗЕ_ ТВ-12мп

Аркушів 20

Київ – 2022

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці

Архітектура та патерни мікрофронтендів, принципи їх реалізації, слабкі та сильні сторни

Виконав: студент 2-го курсу НН ІАТЕ, групи ТВ-12мп Веретьонкін Олексій Сергійович

Керівник: к.т.н., доцент, викладач кафедри ІПЗЕ Гагарін Олександр Олександрович

АКТУАЛЬНІСТЬ

- Розділення великої системи на підсистеми
- Збільшення швидкості розгортання застосунку
- Окрема зона відповідальності для команди у проекті
- Централізована система документообігу та звітності
- Моніторинг енергоспоживання

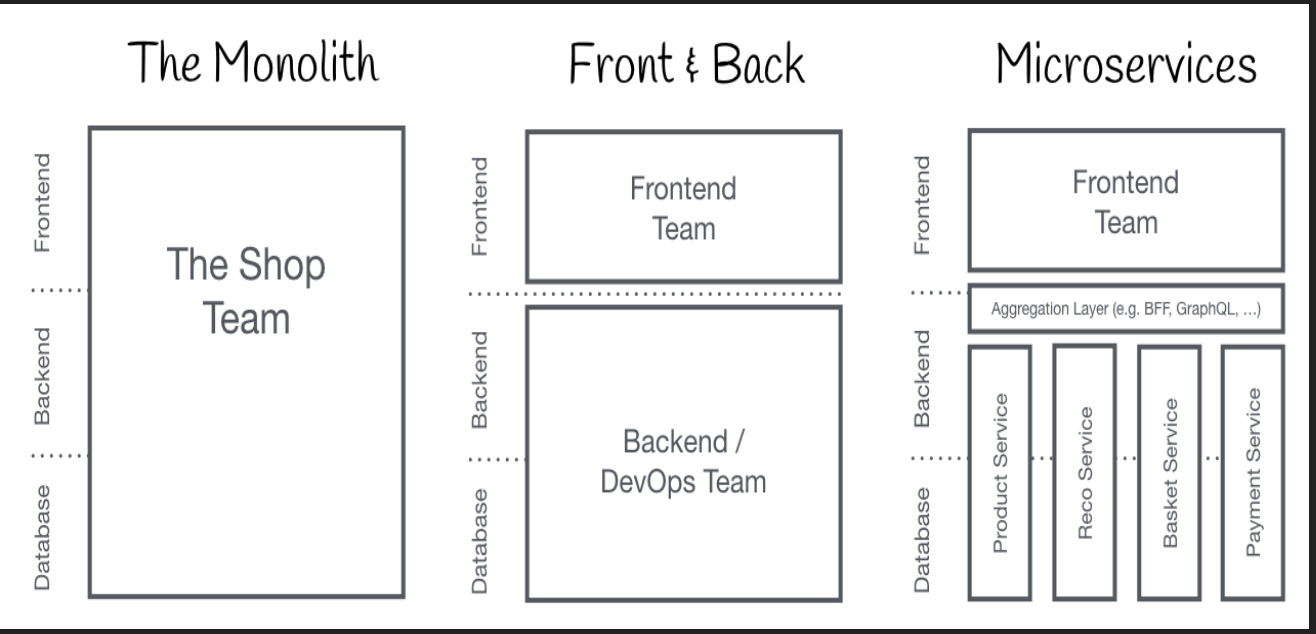
Мета роботи, об’єкт та предмет дослідження

- **Мета роботи:** Розробити веб-застосунок з можливістю подальшого розширення. Розбити його на окремі підсистеми з чітко визначеними межами.
- **Об’єкт дослідження:** Процес розробки фронтенд застосунків
- **Предмет дослідження:** система мікрофронтендів як однин застосунок для управління кабінетом енергоменеджера.

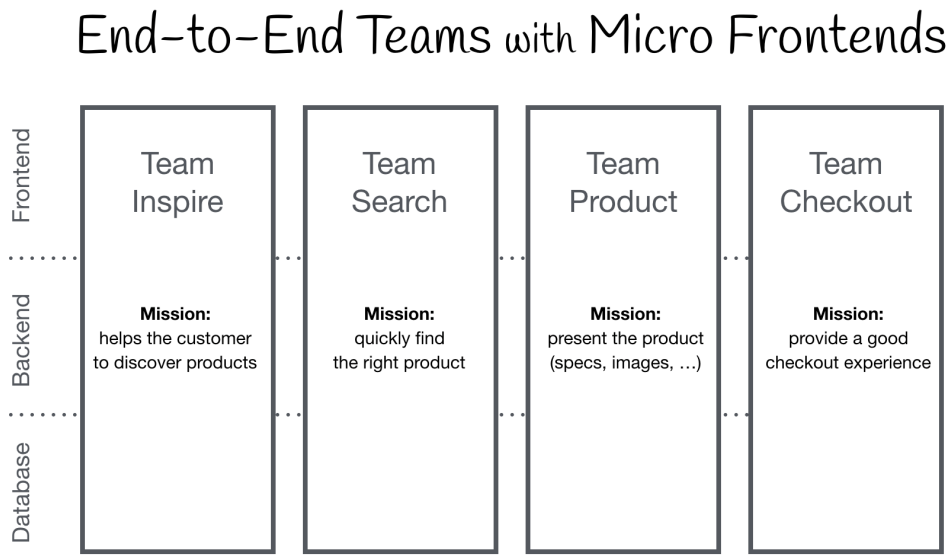
Основні принципи мікрофронтендів

- Незалежність команд що працюють проекті одна від одної
- Ізольованість коду команди
- Узгодження конвенцій між командами
- Використання вбудованих функцій браузера замість побудови власних механізмів комунікації(де це можливо)
- Стійкий застосунок

Проблеми фронтенд монолітів



Моноліт



Мікрофронтенд

Переваги та недоліки

\+

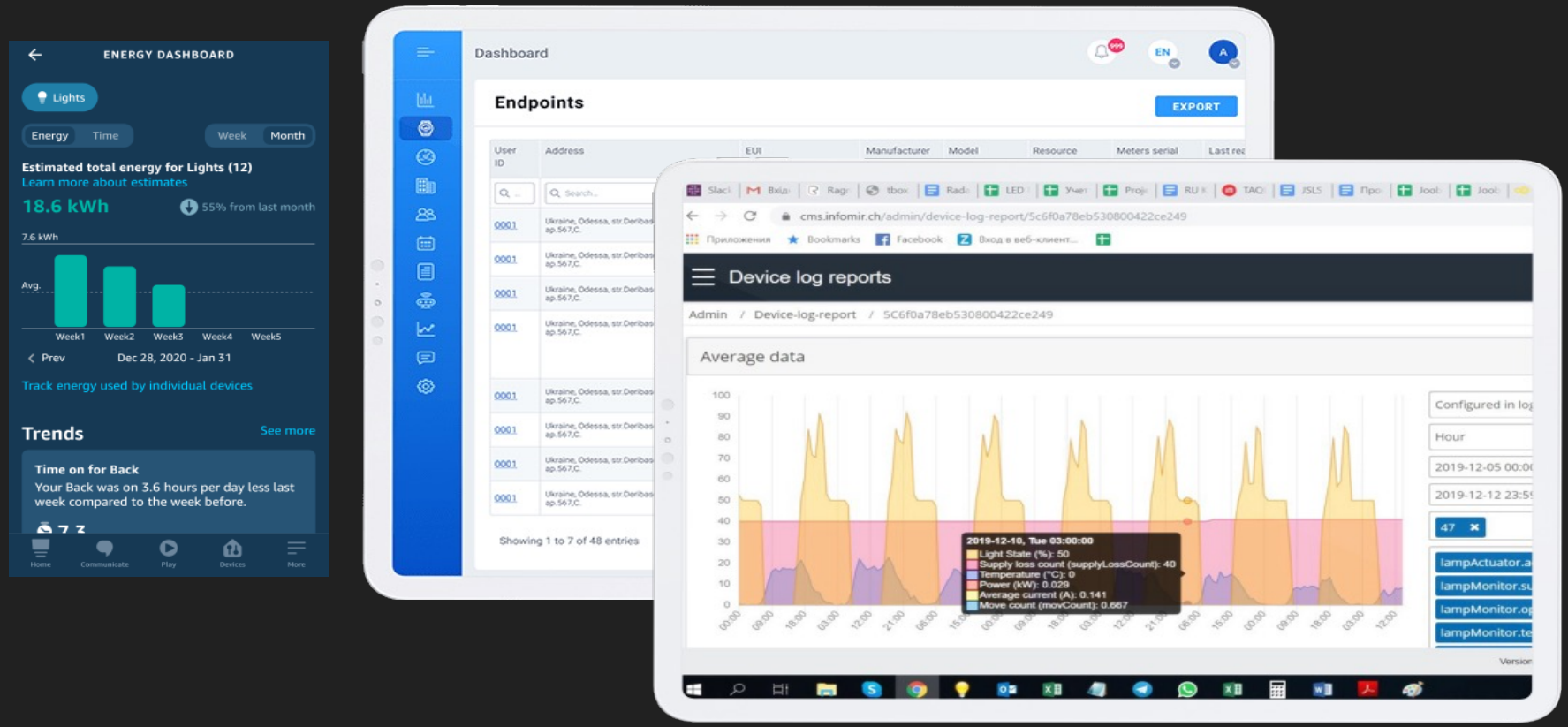
- Розробка бібліотек
- Гнучкість і варіація
- Швидкість
- Масштабованість
- Процес розробки
- Незалежне розгортання та тестування

\-

- Ресурси
- Складність
- Завантаження зайвих ресурсів

Аналогічні системи

- Jooby RDC Dashboard
- Jooby CMS
- Alexa Energy Dashboard



Відомі веб-застосунку на основі мікрофронтендів

https://www.netflix.com/ua/

- Фреймворк React, Webpack 5 Module Federation

https://accounts.spotify.com/en/login

- Iframes(desktop aplication)

Основні технології

TypeScript

Angular

RxJs

NgRx

NgxBootsrap

Webpack 5

React

Розділи застосунку

Сторінка авторизації	Профіль користувача	Головна панель	Панель керування
• Авторизація • Реєстрація	• Редагування даних користувача • Фото користувача • Видалення користувача	• Повідомлення системи • Ведення документообігу	• Користувачі • Будівлі • Локації • Групи • Лічильники

Мікрофронтенди що утворюють систему

Shell

- Початкова точка системи
- Збирає всі мікрофронтенди
- Виконує авторизацію користувача

Portal

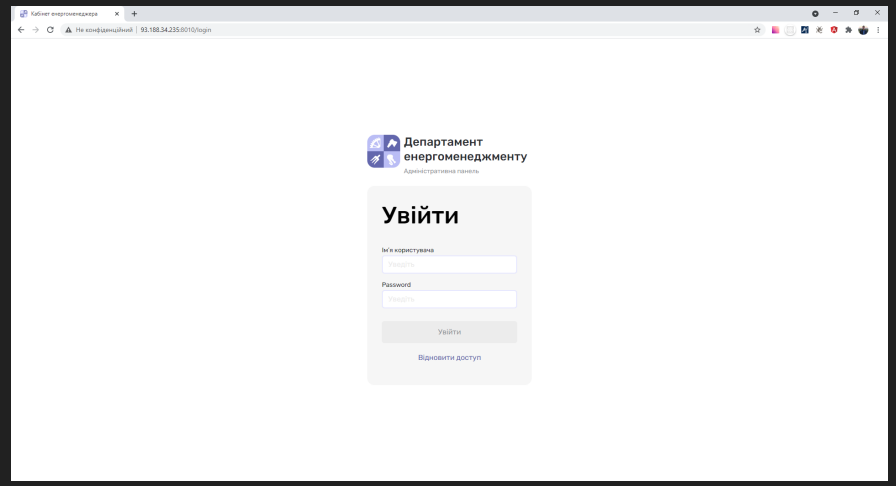
- Панель управління енергоменеджера
- Профіль користувача

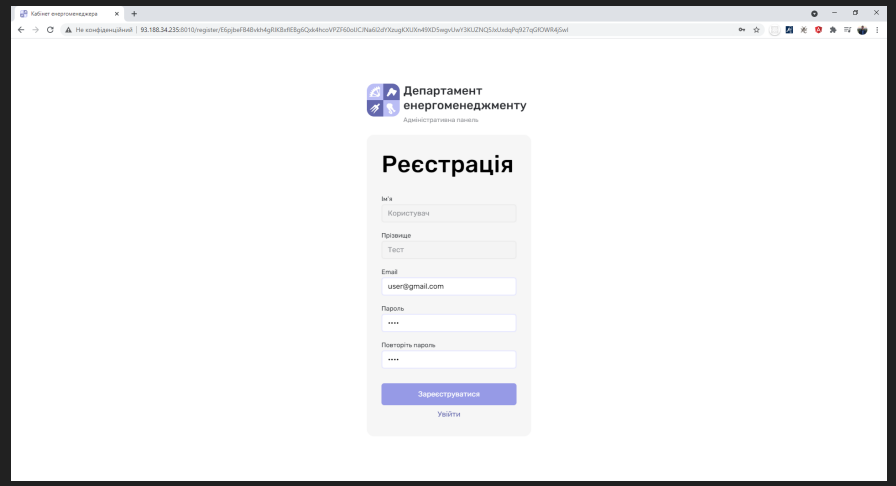
React

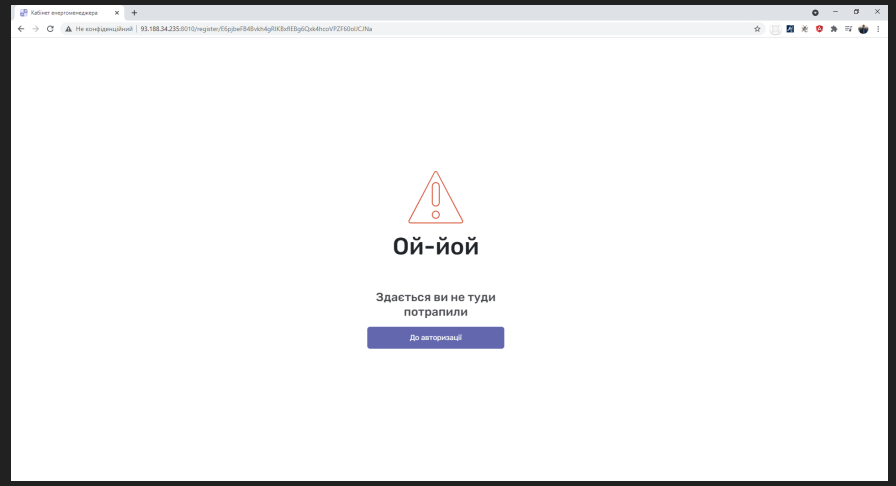
- Мікрофронтенд на базі React
- Показує можливості взаємодії мікрофронтендів

Мікрофронтенд Shell: Сторінка авторизації

- Вхід в систему
- Реєстрація за посиланням



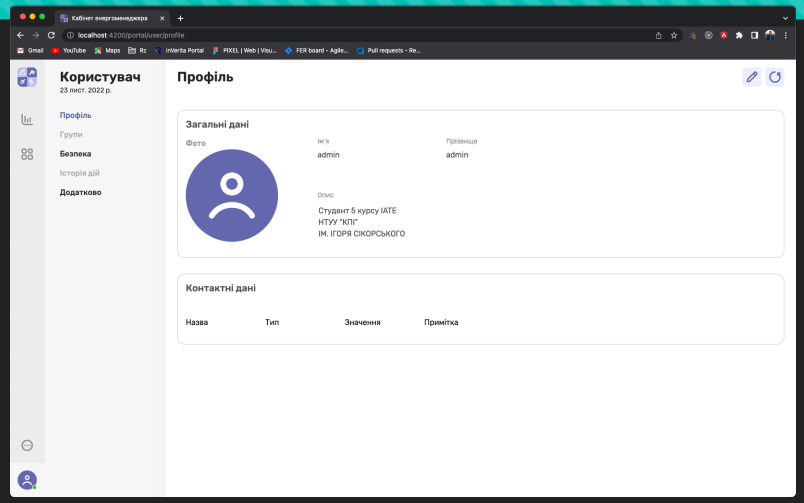


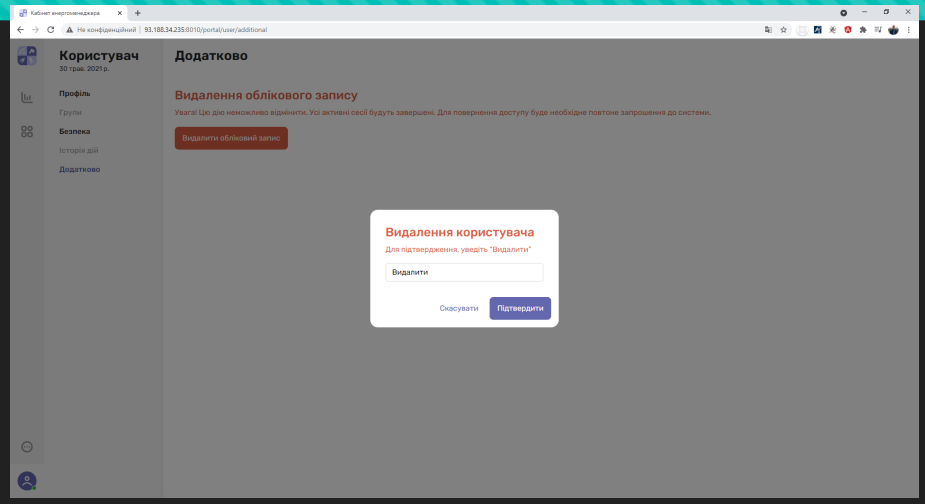


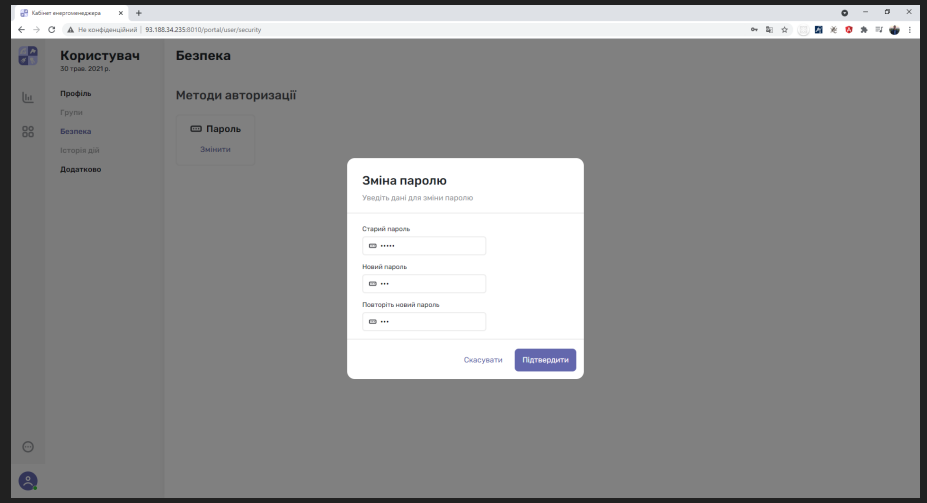
12

Мікрофронтенд Portal: Профіль користувача

- Загальна інформація
- Зміна паролю
- Видалення користувача

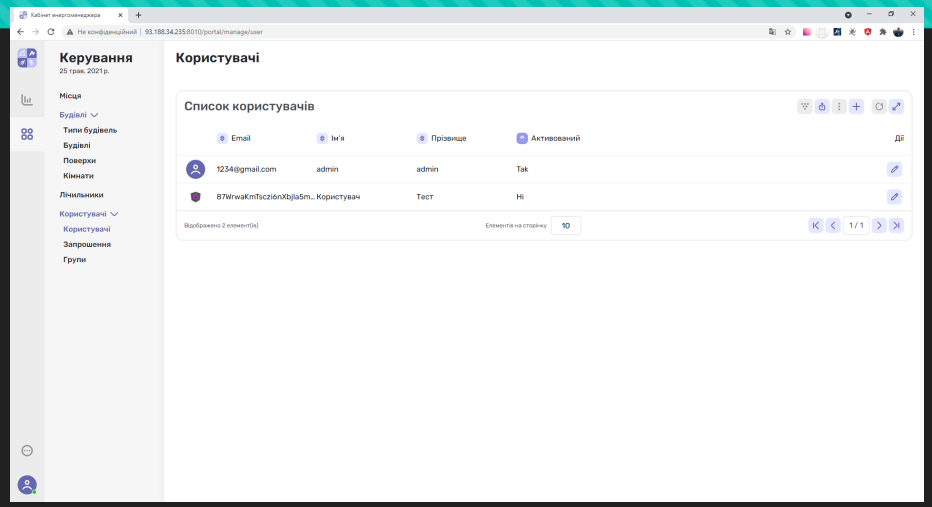


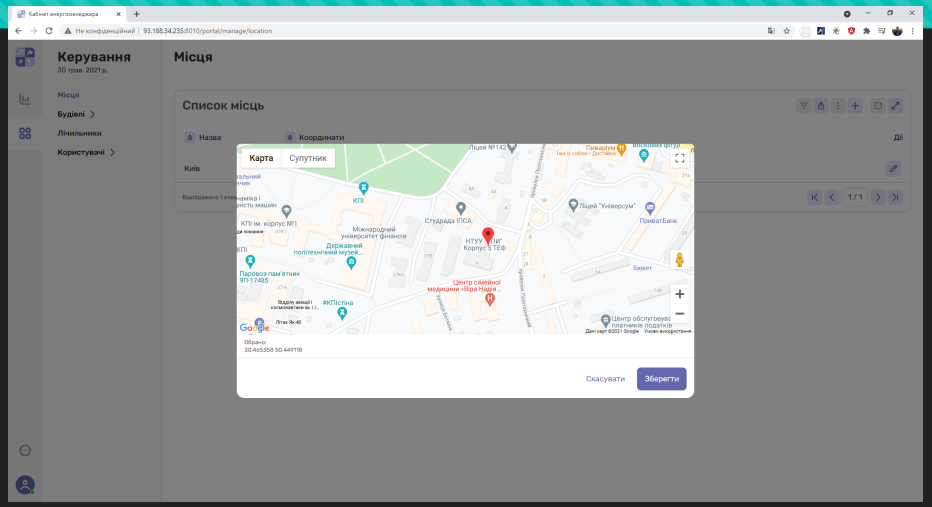


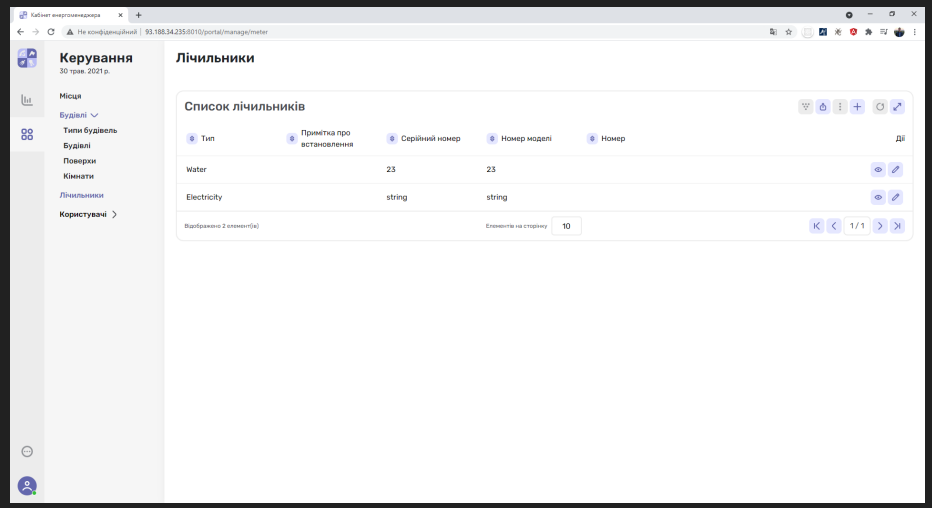


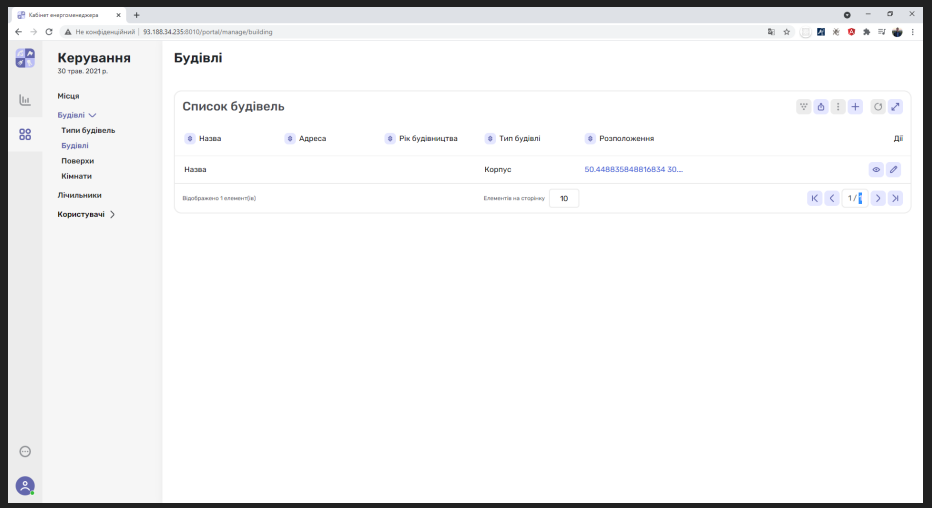
Мікрофронтенд Portal: Панель керування

- Редагування/створення користувачів, будівель, лічильників, локацій
- Сторінки конкретних поверхів, будівель, кімнат

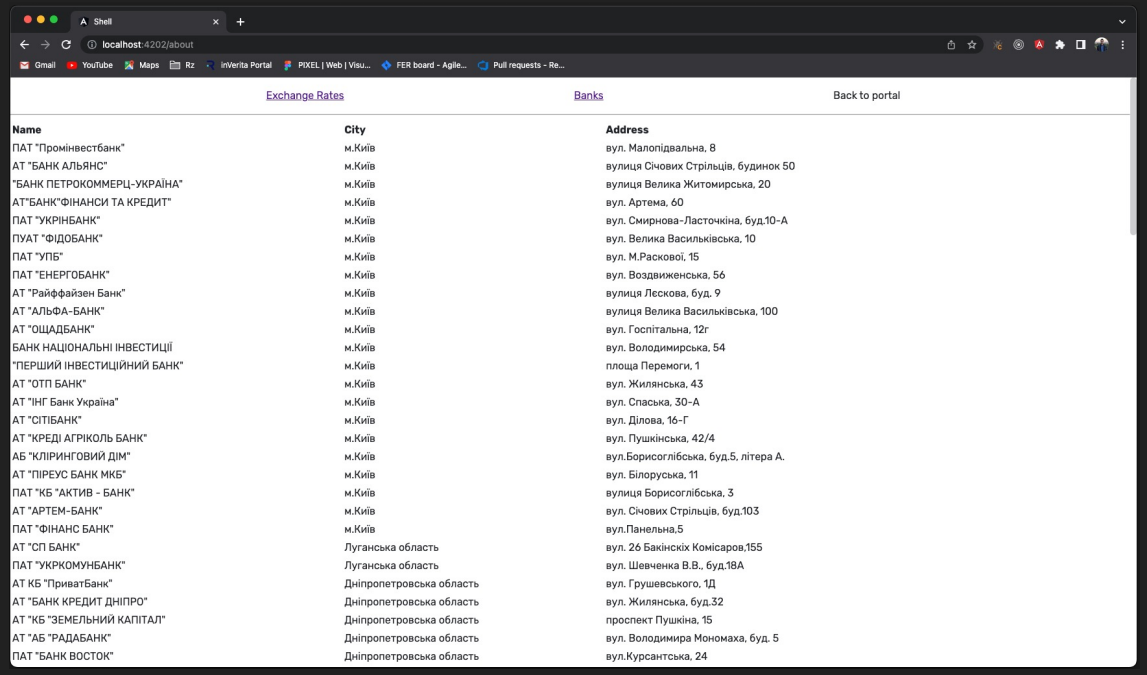




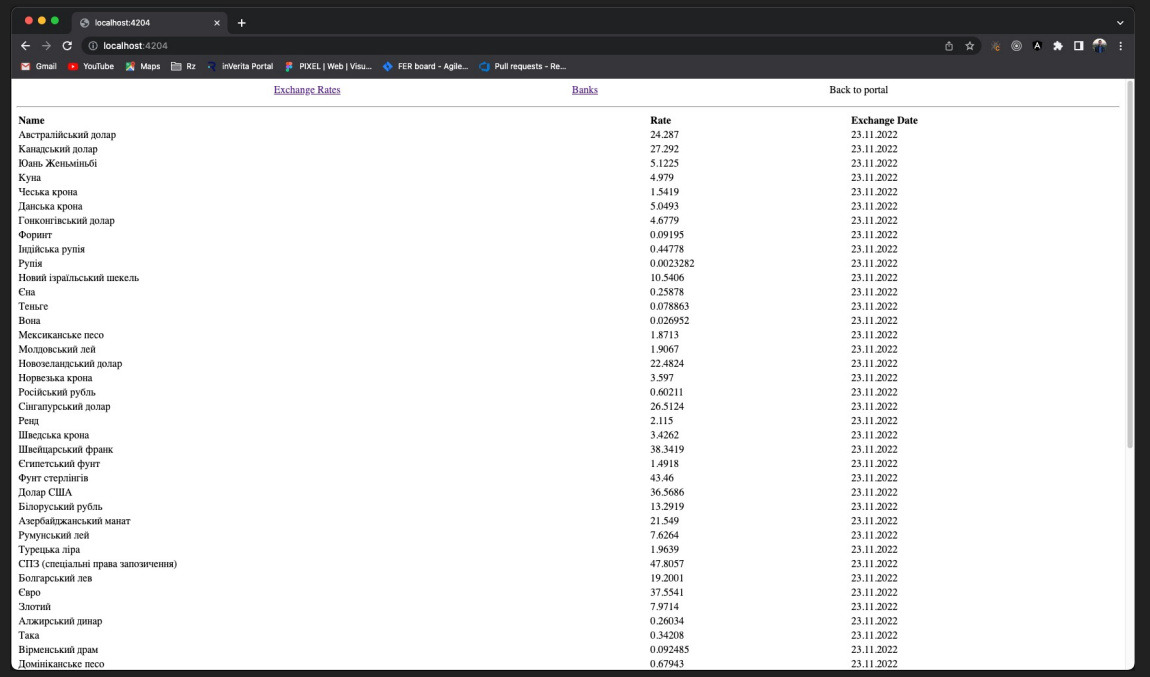




Мікрофронтенд React



Name	City	Address
ПАТ "Промінвестбанк"	м.Київ	вул. Малопідвальна, 8
АТ "БАНК АЛЬЯНС"	м.Київ	вулиця Січових Стрільців, будинок 50
"БАНК ПЕТРОКОММЕРЦ-УКРАЇНА"	м.Київ	вулиця Велика Житомирська, 20
АТ"БАНК"ФІНАНСИ ТА КРЕДИТ"	м.Київ	вул. Артема, 60
ПАТ "УКРІНБАНК"	м.Київ	вул. Смирнова-Ласточкіна, буд.10-А
ПУАТ "ФІДОБАНК"	м.Київ	вул. Велика Васильківська, 10
ПАТ "УПБ"	м.Київ	вул. М.Раскової, 15
ПАТ "ЕНЕРГОБАНК"	м.Київ	вул. Воздвиженська, 56
АТ "Райффайзен Банк"	м.Київ	вулиця Лескова, буд. 9
АТ "АЛЬФА-БАНК"	м.Київ	вулиця Велика Васильківська, 100
АТ "ОЩАДБАНК"	м.Київ	вул. Госпітальна, 12г
БАНК НАЦІОНАЛЬНІ ІНВЕСТИЦІЇ	м.Київ	вул. Володимирська, 54
"ПЕРШИЙ ІНВЕСТИЦІЙНИЙ БАНК"	м.Київ	площа Перемоги, 1
АТ "ОТП БАНК"	м.Київ	вул. Жилянська, 43
АТ "ІНГ Банк Україна"	м.Київ	вул. Спаська, 30-А
АТ "СІТІБАНК"	м.Київ	вул. Ділова, 16-Г
АТ "КРЕДІ АГРІКОЛЬ БАНК"	м.Київ	вул. Пушкінська, 42/4
АБ "КЛІРИНГОВИЙ ДІМ"	м.Київ	вул.Борисоглібська, буд.5, літера А.
АТ "ПІРЕУС БАНК МКБ"	м.Київ	вул. Білоруська, 11
ПАТ "КБ "АКТИВ - БАНК"	м.Київ	вулиця Борисоглібська, 3
АТ "АРТЕМ-БАНК"	м.Київ	вул. Січових Стрільців, буд.103
ПАТ "ФІНАНС БАНК"	м.Київ	вул.Панельна,5
АТ "СП БАНК"	Луганська область	вул. 26 Бакінських Комісаров,155
ПАТ "УКРКОМУНБАНК"	Луганська область	вул. Шевченка В.В., буд.18А
АТ КБ "ПриватБанк"	Дніпропетровська область	вул. Грушевського, 1Д
АТ "БАНК КРЕДИТ ДНІПРО"	Дніпропетровська область	вул. Жилянська, буд.32
АТ "КБ "ЗЕМЕЛЬНИЙ КАПІТАЛ"	Дніпропетровська область	проспект Пушкіна, 15
АТ "АБ "РАДАБАНК"	Дніпропетровська область	вул. Володимира Мономаха, буд. 5
ПАТ "БАНК ВОСТОК"	Дніпропетровська область	вул.Курсантська, 24

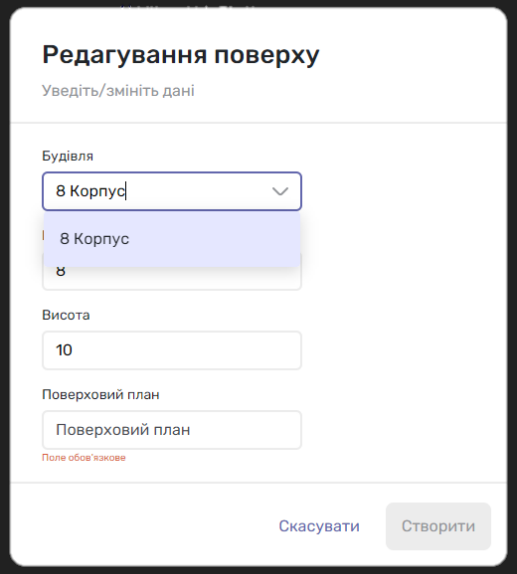


Name	Rate	Exchange Date
Австралійський долар	24.287	23.11.2022
Канадський долар	27.292	23.11.2022
Юань Женьміньбі	5.1225	23.11.2022
Куна	4.979	23.11.2022
Чеська крона	1.5419	23.11.2022
Данська крона	5.0493	23.11.2022
Гонконгівський долар	4.6779	23.11.2022
Форинт	0.09195	23.11.2022
Індійська рупія	0.44778	23.11.2022
Рупія	0.0023282	23.11.2022
Новий ізраїльський шекель	10.5406	23.11.2022
Єна	0.25878	23.11.2022
Тенге	0.078863	23.11.2022
Вона	0.026952	23.11.2022
Мексиканське песо	1.8713	23.11.2022
Молдовський лей	1.9067	23.11.2022
Новозеландський долар	22.4824	23.11.2022
Норвезька крона	3.597	23.11.2022
Російський рубль	0.60211	23.11.2022
Сінгапурський долар	26.5124	23.11.2022
Ренд	2.115	23.11.2022
Шведська крона	3.4262	23.11.2022
Швейцарський франк	38.3419	23.11.2022
Єгипетський фунт	1.4918	23.11.2022
Фунт стерлінгів	43.46	23.11.2022
Долар США	36.5686	23.11.2022
Білоруський рубль	13.2919	23.11.2022
Азербайджанський манат	21.549	23.11.2022
Румунський лей	7.6264	23.11.2022
Турецька ліра	1.9639	23.11.2022
СПЗ (спеціальні права запозичення)	47.8057	23.11.2022
Болгарський лев	19.2001	23.11.2022
Євро	37.5541	23.11.2022
Злотий	7.9714	23.11.2022
Алжирський динар	0.26034	23.11.2022
Така	0.34208	23.11.2022
Вірменський драм	0.092485	23.11.2022
Домініканське песо	0.67943	23.11.2022

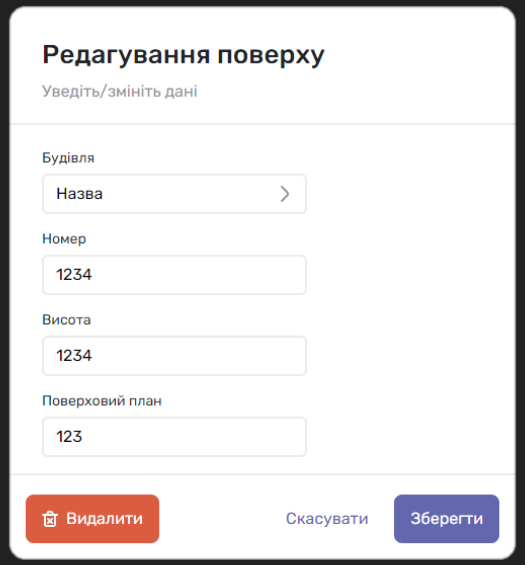
Мікрофронтенд React поєднується з Shell

Мікрофронтенд React як окремий застосунок

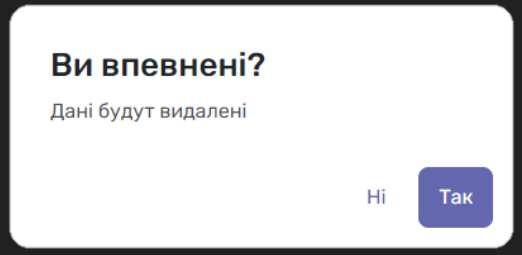
Бібліотека компонентів: Модальні вікна



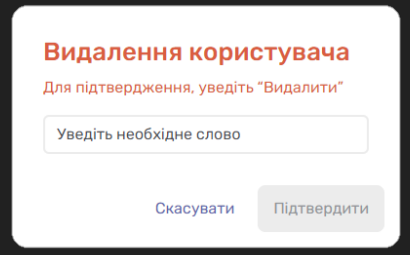
Створення нової сутності



Редагування сутності

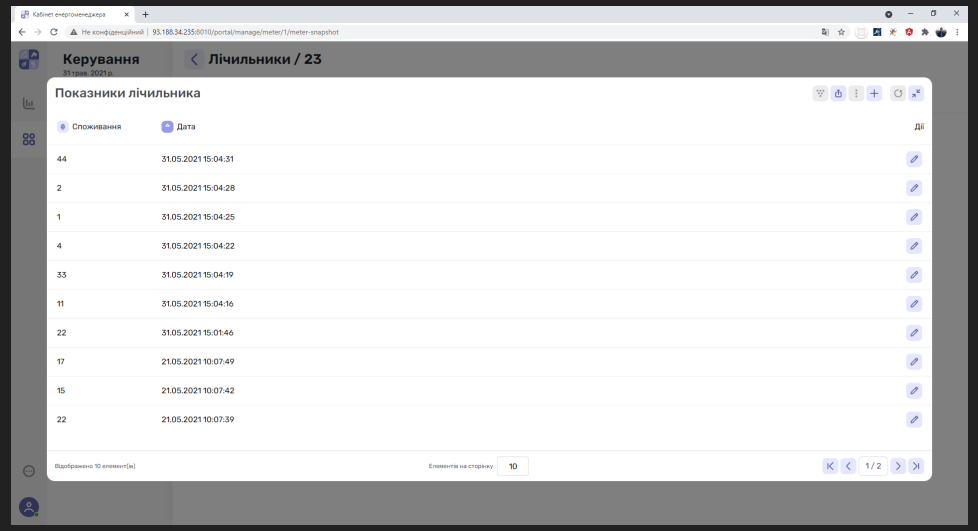


Вибір так або ні

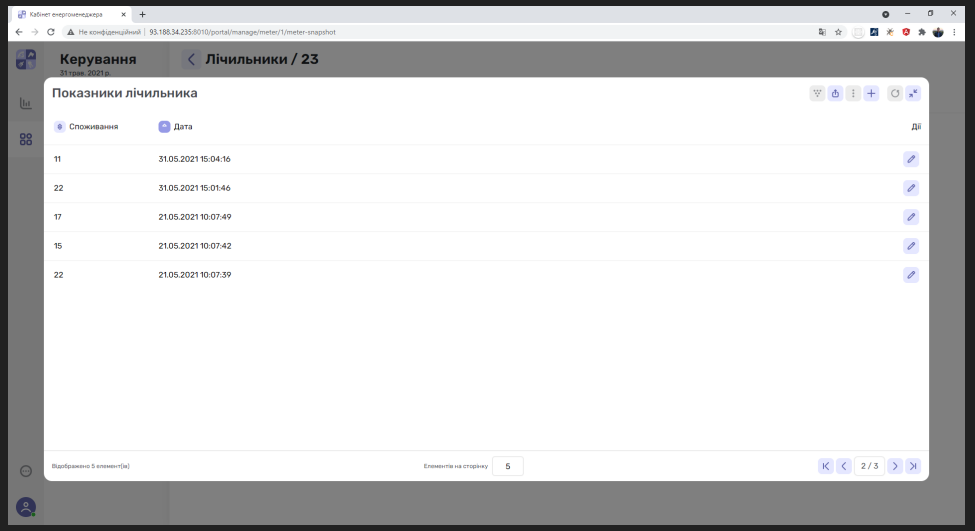


Видалення

Бібліотека компонентів: Таблиця

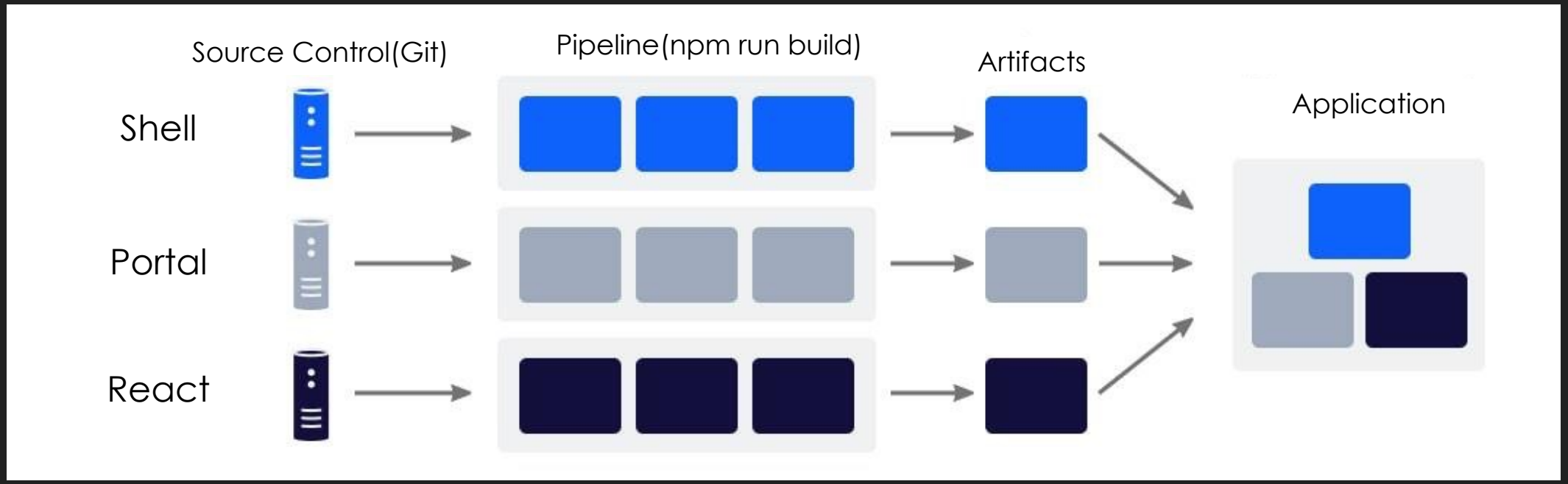


Сортування за датою по спаданню



Елементів на сторінку: 5,
Загальна к-сть сторінок: 3,
Обрана сторінка: 2

Схема розгортання системи



Висновки

- Було розроблено гнучку систему, що задовольняє початковим потребам.
- Розглянуто основні патерни побудови мікрофронтендів: SSR, CSR
- Застосунок складається з трьох мікрофронтендів
- Зазначено слабкі та сильні сторони мікрофронтендної архітектури
- Визначено основні підходи для реалізації мікрофрнонендів
- Мікрофронтенди використовують спільну бібліотеку, що реалізує найбільш пере використовувані компоненти
- Для оновлення системи достатньо лише розгорнути оновлений мікрофронтенд, а не всю систему.
- Можливі покращення:
  - CI/CD
  - Розбити мікфрофронтенд Portal на більш малі мікрофронтенди
  - Додати SSR(server side rendering)

Дякую за увагу!