Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКУ ДЛЯ СПОЖИВАЧІВ З ВИКОРИСТАННЯМ МІКРОСЕРВІСІВ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-18-1

Ісмоілов Р.М.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Кобилін О.А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Ісмоілову Рустаму Музафарджоновичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення вебзастосунку для споживачів з використанням мікросервісів

затверджена наказом університету від 16 травня 2022 року № 541Ст

2. Термін подання студентом роботи до екзаменаційної комісії 28 травня 2022 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, React, Django, Python, системи управління реляційними базами даних PostgreSQL, атомарний дизайн розробки вебзастосунків з використанням мікросервісів та мікросервісної архітектури.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз предметної галузі, аналіз аналогічних проєктів;
2. Розробка технічного завдання;
3. Розробка архітектури веб-ресурсу;
4. Розробка серверної частини, розробка адміністраторської частини, розробка клієнтської частини

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) лістинг коду, мета роботи, постановка задачі, етапи виконання роботи, результати роботи.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	Доцент Бєлова Н.В.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	18.04.2022	
2	Аналіз завдання, підбір літератури	18.04.22-21.04.22	
3	Аналіз літератури з досліджуваної проблеми	22.04.22-25.04.22	
4	Аналіз технічних засобів	26.04.22-30.04.22	
5	Проєктування бази даних	01.05.22-14.05.22	
6	Програмна реалізація	15.05.22-23.05.22	
7	Оформлення пояснювальної записки	24.05.22-28.05.22	
8	Перевірка на плагіат	31.05.22	
9	Рецензування	1.06.22	
10	Підготовка презентації та доповіді	2.06.22-05.06.22	
11	Занесення роботи в електронний архів	06.06.22	
12	Попередній захист кваліфікаційної роботи	09.06.22	

Дата видачі завдання 18 квітня 2022 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Кобилін О.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 44 с., 21 рис., 34 джерело.

REACT, PYTHON, ВЕБЗАСТОСУНОК, I18N, МІКРОСЕРВІСНА АРХІТЕКТУРА, DEPENDENCY INJECTION, БАЗА ДАНИХ, ПЕРЕДАЧА ОБ'ЄКТІВ У ФОРМАТІ JSON.

Об'єктом роботи є створення вебзастосунку для заохочення клієнтів в різні ресурси.

Метою роботи є розроблення вебзастосунку для просування компаній на різних платформах. Використано SOLID принципи програмування, методи мікросервісної архітектури, методологію поділу будь−якого інтерфейсу на функціональні компоненти Atomic Design, інтернаціоналізацію i18n, фреймворк React, основні методи передачі даних між клієнтом та сервером.

У результаті роботи здійснена програмна реалізація готового вебзастосунку для споживача.

REACT, PYTHON, WEB APP, I18N, MICROSERVICE ARCHITECTURE, DEPENDENCY INJECTION, DATABASE, TRANSFER OF OBJECTS IN JSON FORMAT.

The object of the work is the creation of a web app for the interest of clients in various resources.

The aim of the work is to develop a web application for promoting companies on different platforms. SOLID programming principles, methods of microservice architecture, methodology of division of any interface into functional components of Atomic Design, i18n internationalization, React framework, basic methods of data transfer between client and server are used.

As a result, the software implementation of the finished web application for the consumer was carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

JSON −JavaScript Object Notation

HTTP −HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

HTML – HyperText Markup Language

AJAX – Asynchronous JavaScript and XML

XML – eXtensible Markup Language

URI – Uniform Resource Identifier

REST – Representational State Transfer

SOAP – Simple Object Access Protocol

API – Application Programming Interface

CQRS – Command−query responsibility segregation

CI/CD – Continuous integration / continuous delivery

SLA – Service−level agreement

JSF – Java Server Faces

JDBC – Java Database Connectivity

ORM – Object−relational mapping

OXM – Object XML Mapping

DI – Dependency Injection

AES – Advanced Encryption Standard

DES – Data Encryption Standard

IDEA – International Data Encryption Algorithm

ВСТУП

Сьогодні програми змінили наше життя. Від покупок і відпочинку до роботи, деякі аспекти нашого життя залишилися незайманими революцією застосунків. Всі люди добре знайомі з мобільними програмами, тому що вони відіграли таку помітну роль у нашому повсякденному житті. Але вебзастосунки однаково поширені і так само важливі.

Вебзастосунок – це свого роду комп'ютерна програма. Вона використовує онлайн технології (включаючи браузери) для виконання величезного спектра різних завдань. Багато програм використовуються для онлайн−рітейлу, проте, вони можуть служити для різних цілей, від замовлення їжі на винос до бронювання свят. Крім того, вебзастосунок може бути чимось таким же простим, як контактні форми вебсайту або онлайн калькулятори.

Вебпрограми мають безліч переваг. Зокрема, вони допомагають скоротити витрати для бізнесу та окремих користувачів. Це пов'язано з тим, що вони вимагають меншого обслуговування, а також можуть мати нижчі вимоги до комп'ютерів користувачів (з точки зору обчислювальної потужності тощо). Це з тим, що обробка ефективно відбувається вс інших місцях.

1 ОГЛЯД ОСНОВНИХ МЕТОДІВ РЕАЛІЗАЦІЇ ЗАСТОСУНКУ

1.1 Сучасні архітектури розробки вебзастосунку

1.1.1 Монолітна архітектура розробки вебзастосунку

Монолітна архітектура представляє собою розробку вебзастосунку як єдине ціле. Тобто якщо розбити вебзастосунок на клієнт та сервер, то це не будуть дві різні частини одного цілого, це можна представити лише як одне ціле. Монолітна розробка полягає в тому що клієнтська частина та серверна будуть виконуватися за допомогою одного і того ж модуля (сервера). Можна сказати, що моноліт це деякий кам'яний блок, тобто він неподільний. Звичайний монолітний вебзастосунок поділяється на базу даних (для зберігання різного типу даних, залежить від предметної області), клієнтського користувацького інтерфейсу (все, що бачить користувач) та серверного застосунку (в ньому прописується вся бізнес-логіка застосунку). Всі ці три частини виконуються в одному місці. Можна наглядно це розглянути на рисунку 1.1 [1].

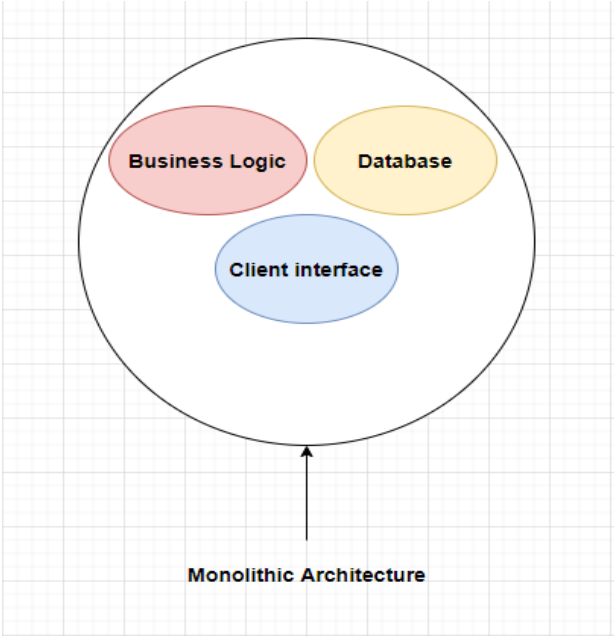

Рисунок 1.1 – Схема монолітної архітектури

Існують два типи монолітної архітектури: монолітна архітектура с одним процесом та модульна монолітна архітектура.

Монолітна архітектура з одним процесом – це коли застосунок працює як єдиний процес, він не розбитий на певну кількість модулів та має зв'язок з однією базою даних [1].

Цей тип підходить для якихось невеликих проєктів, наприклад стартапів, які тільки розвиваються або якийсь тренувальний проєкт. Приклад монолітної архітектури з одним процесом можна розглянути на рис. 1.2 [1].

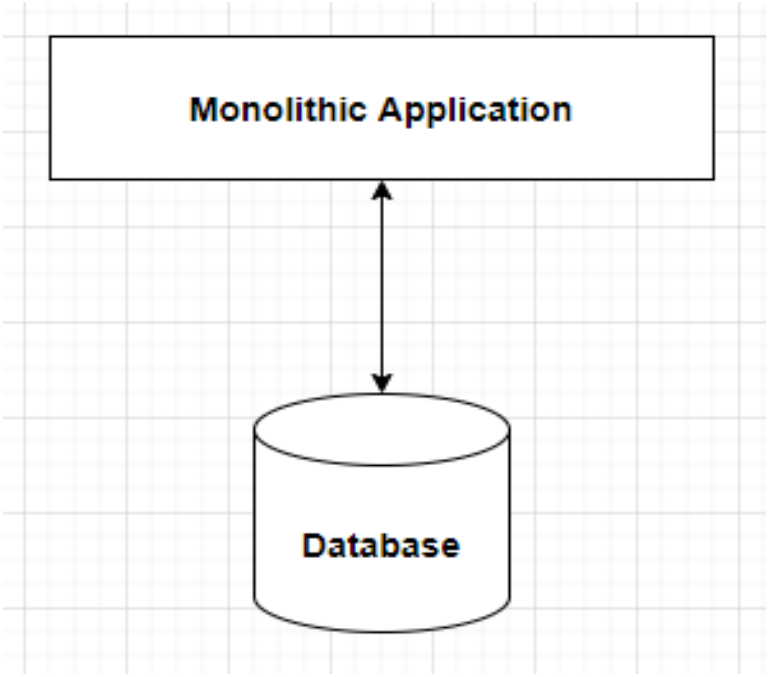

Рисунок 1.2 – Структура монолітної архітектури з одним процесом

Модульна монолітна архітектура – це означає, що процес розбитий на певну кількість модулів, тобто кожний із цих модулів може працювати окремо, але вони поєднанні в один файл та працюють від однієї бази даних. Також ці модулі пов'язуються між собою [1].

Модульна монолітна архітектура вважається кращою чим монолітна архітектура с одним процесом. Приклад модульної монолітної архітектури можна розглянути на рисунку 1.3 [1].

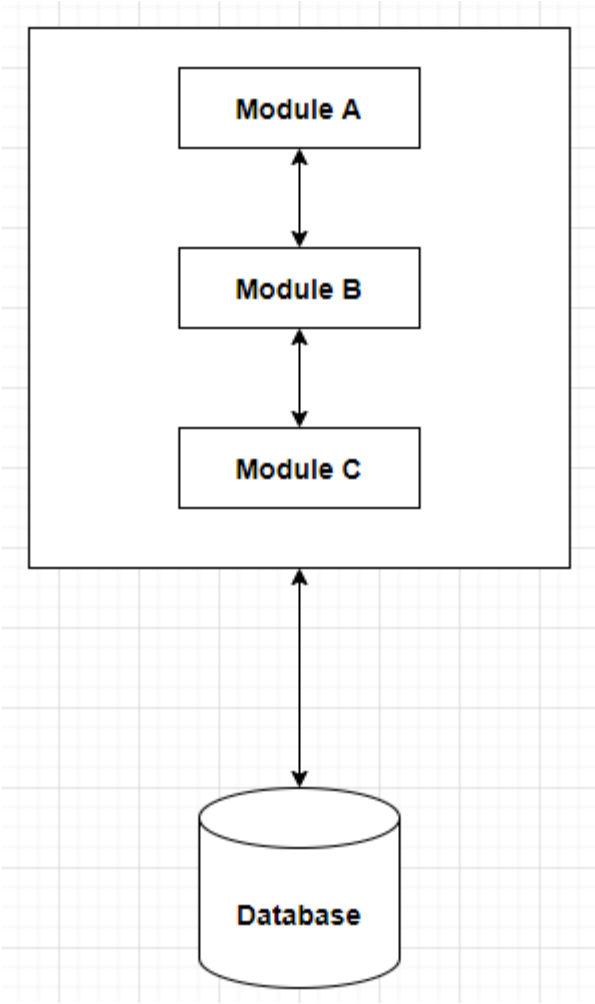

Рисунок 1.3 – Структура модульної монолітної архітектури

Монолітна архітектура широко популярна серед починаючих програмістів або якщо це доволі невеликий проєкт, який не потребує великої логіки, наприклад сайти інформаційного типу. Насамперед, що треба розуміти використовуючи монолітну архітектуру, це те, що всі компоненти в монолітному застосунку є взаємопов’язані та взаємозалежні, що дозволяє монолітному застосунку бути самодостатнім. Хоча серед розробників є популярна думка, що монолітна архітектура являється застарілою, але можливо в деяких специфічних ситуаціях – це гарне рішення для вирішення технічного завдання.

1.1.2 Вебсервіси, SOAP архітектура розробки вебзастосунку

Вебсервіс − це набір відкритих протоколів та стандартів, які дозволяють обмінюватися даними між різними програмами або системами. Вебсервіси можуть використовуватися програмами, написаними різними мовами програмування та працюючими на різних платформах, для обміну даними через комп'ютерні мережі, такі як Інтернет, аналогічно між процесного зв'язку на одному комп'ютері.

Будь−яке програмне забезпечення, програма або хмарна технологія, що використовує стандартизовані вебпротоколи (HTTP або HTTPS) для підключення, взаємодії та обміну повідомленнями даних − зазвичай XML (розширювана мова розмітки) − через Інтернет вважається вебсервісом.

Перевага вебсервісів полягає в тому, що програми, розроблені різними мовами, можуть зв'язуватися один з одним шляхом обміну даними через вебсервіс між клієнтами та серверами. Клієнт викликає вебслужбу, відправляючи запит XML, на який служба відповідає XML−відповіддю.

На наведеній нижче діаграмі (рис. 1.4) показано дуже спрощене уявлення про те, як буде працювати вебслужба. Клієнт буде викликати серію дзвінків вебслужби через запити до сервера, на якому буде розміщено фактичну вебслужбу.

Ці запити виконуються за допомогою так званого віддаленого виклику процедури. Видалені дзвінки процедур (RPC) − це дзвінки методів, розміщених відповідною вебслужбою.

Основним компонентом дизайну вебсервісу є дані, що передаються між клієнтом та сервером, а саме XML. XML (розширювана мова розмітки) є аналогом HTML і проміжною мовою, що легко розуміється, який розуміється багатьма мовами програмування [10].

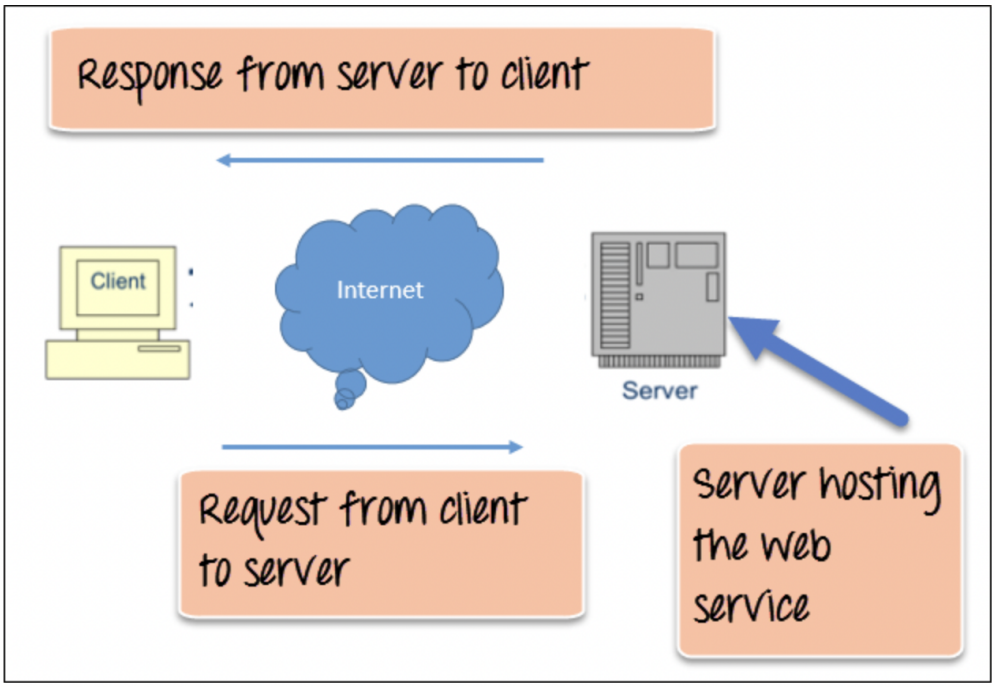

Рисунок 1.4 – Діаграма роботи вебсервісу

Тому, коли програми розмовляють одна з одною, вони насправді розмовляють у XML. Це забезпечує спільну платформу для програм, розроблених різними мовами програмування, щоб спілкуватися один з одним.

Вебслужби використовують так званий SOAP (Simple Object Access Protocol) для надсилання даних XML між програмами. Дані надсилаються за звичайним HTTP. Дані, що надсилаються з вебслужби до програми, називаються повідомленнями SOAP. Повідомлення SOAP − це нічим іншим, як XML−документ. Оскільки документ написаний на XML, клієнтська програма, що викликає вебслужбу, може бути написана будь−якою мовою програмування.

SOAP відомий як протокол доступу до простих об’єктів, але в більш пізні часи він був скорочений до SOAP v1.2. SOAP − це протокол або, тобто, визначення того, як вебсервіси взаємо згадують один з одним, або клієнтські програми, які їх викликають.

SOAP був розроблений як проміжна мова, щоб програми, побудовані різними мовами програмування, могли легко спілкуватися один з одним та уникати екстремальних зусиль з розробки (рис. 1.5) [10].

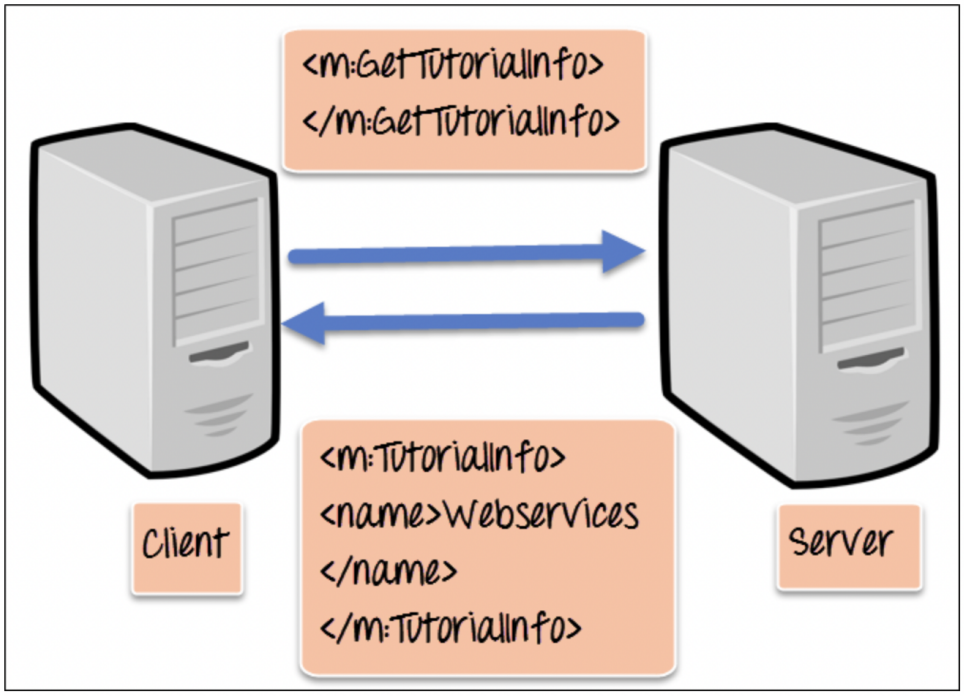

Рисунок 1.5 – Приклад зв’язку через SOAP

SOAP − це протокол, який використовується для обміну даними між програмами. Нижче наведено деякі з причин використання SOAP:

– під час створення вебсервісів на основі SOAP необхідно мати деякі мови, які можна використовувати для вебсервісів для спілкування з клієнтськими програмами. SOAP − це ідеальне середовище, яке було розроблено для досягнення цієї мети. Цей протокол також рекомендується консорціумом W3C, який є керівним органом для всіх вебстандартів;

– SOAP − це легкий протокол, який використовується для обміну даними між програмами. Зверніть увагу на ключове слово «light». Оскільки програмування SOAP засноване на мові XML, яка сама по собі є легкою мовою обміну даними, тому SOAP як протокол, який також відноситься до тієї ж категорії;

– SOAP розроблено так, щоб бути незалежним від платформи, а також не залежить від операційної системи. Таким чином, протокол SOAP може працювати з будь–якими програмами на основі мови програмування як на Windows, так і на Linux;

– він працює на протоколі HTTP – SOAP працює на протоколі HTTP, який є протоколом за промовчанням, що використовується всіма вебпрограмами. Отже, немає жодної настройки, яка не потрібна для запуску вебсервісів, побудованих на протоколі SOAP, для роботи у Всесвітній мережі.

Повідомлення SOAP зазвичай автоматично генеруються вебслужбою під час його виклику.

Щоразу, коли клієнтська програма викликає метод у вебслужбу, вебслужба автоматично генерує SOAP–повідомлення, яке матиме необхідну інформацію про дані, які будуть надіслані з вебслужби до клієнтської програми.

З рисунка 1.6 бачимо, що перша частина повідомлення SOAP – це елемент конверта, який використовується для інкапсуляції всього SOAP–повідомлення. Наступним елементом є тіло SOAP, яке містить докладну інформацію про фактичне повідомлення. Повідомлення містить вебсервіс, який має назву Guru99WebService [10].

Рисунок 1.6 – Структура повідомлень SOAP

«Guru99Webservice» приймає параметр типу «int» і має ім’я TutorialID. Тепер вищезазначене повідомлення SOAP буде надіслано між вебслужбою та клієнтською програмою.

1.1.3 Мікросервіси

Мікросервіси – це архітектурний та організаційний підхід до розробки програмного забезпечення, де програмне забезпечення складається з невеликих незалежних сервісів, що взаємодіють через чітко визначені API. Ці послуги належать невеликим автономним командам.

Архітектури мікросервісів спрощують масштабування та прискорюють розробку застосунків, забезпечуючи інновації та прискорюючи час виходу на ринок нових функцій. Вона складається із колекції невеликих автономних сервісів. Це продемонстровано на рисунку 1.7. Кожна послуга є автономною та має реалізовувати єдину бізнес–можливість в обмеженому контексті. Обмежений контекст є природним поділом усередині бізнесу та забезпечує явний кордон, в якому існує доменна модель [2].

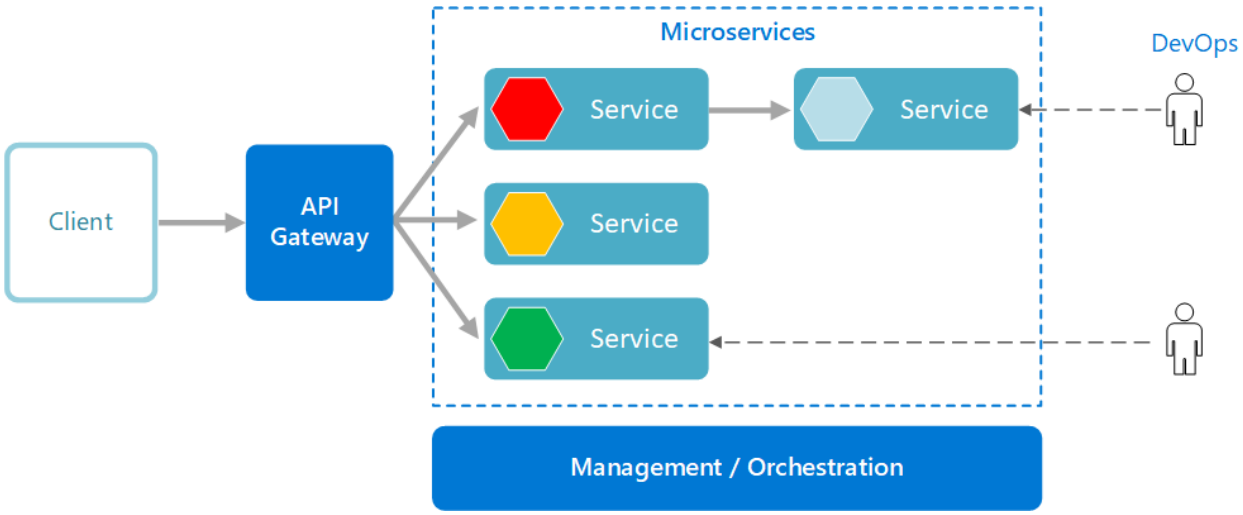

Рисунок 1.7 – Архітектура мікросервісів

Завдяки монолітним архітектурам всі процеси тісно пов’язані та працюють як єдиний сервіс. Це означає, що якщо один процес застосування відчуває сплеск попиту, вся архітектура має бути масштабована. Додавання або поліпшення функцій монолітної програми стає все більш складним у міру зростання кодової бази. Ця складність обмежує експерименти та ускладнює реалізацію нових ідей. Монолітні архітектури підвищують ризик доступності зас, тому що багато залежних та тісно пов’язаних процесів збільшують вплив збою одного процесу (рис. 1.8) [2].

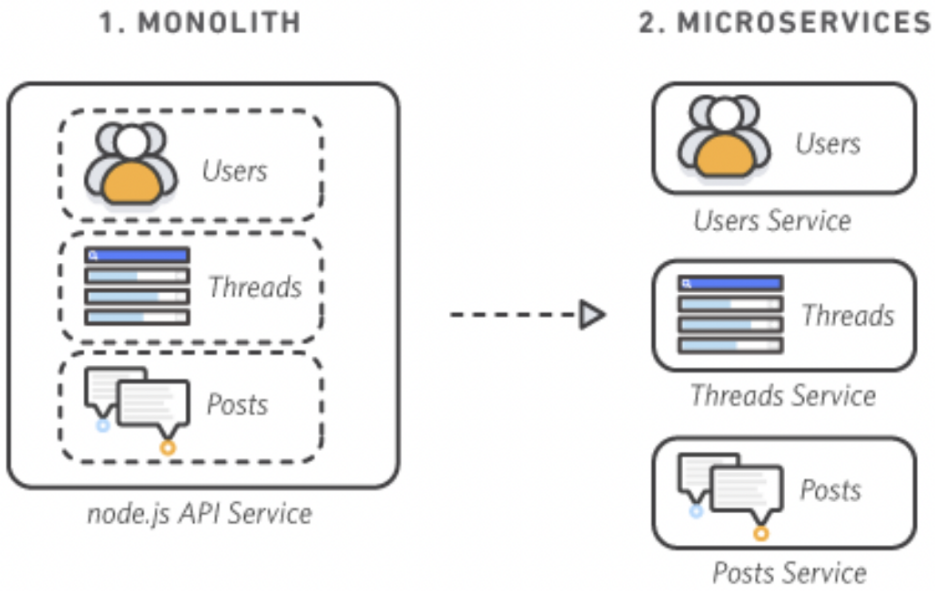

Рисунок 1.8 – Руйнування монолітної програми на мікросервіси

З архітектурою мікросервісів програма створюється як незалежні компоненти, які запускають кожен процес програми як послугу. Ці служби взаємодіють через чітко визначений інтерфейс за допомогою полегшених API. Послуги створені для бізнес−можливостей і кожна служба виконує одну функцію. Оскільки вони працюють незалежно, кожна служба може бути оновлена, розгорнута та масштабована для задоволення попиту на конкретні функції програми.

Переваги мікросервісів:

− гнучкість. Мікросервіси сприяють організації невеликих незалежних команд, які беруть він відповідальність за послуги. Команди діють у невеликому та добре зрозумілому контексті та мають можливість працювати більш незалежно та швидше. Це скорочує час циклу розробки;

− гнучке масштабування. Мікросервіси дозволяють незалежно масштабувати кожну службу відповідно до попиту на функцію програми, що нею підтримується. Це дозволяє командам визначати потреби в інфраструктурі правильного розміру, точно вимірювати вартість функції та підтримувати доступність, якщо послуга зазнає сплеску попиту;

− просте розгортання. Мікросервіси забезпечують безперервну інтеграцію та безперервну доставку, що дозволяє легко випробувати нові ідеї та відкотити, якщо щось не спрацює. Низька вартість збою дозволяє експериментувати, полегшує оновлення коду та прискорює час виходу на ринок нових функцій;

− технологічна свобода. Архітектури мікросервісів не слідують підходу «один розмір підходить усім». Команди мають свободу вибору найкращого інструменту на вирішення своїх конкретних проблем. Як наслідок, команди, що створюють мікросервіси, можуть вибрати найкращий інструмент для кожної роботи;

− багаторазовий код. Поділ програмного забезпечення на невеликі, чітко визначені модулі дозволяє командам використовувати функції для кількох цілей. Служба, написана для певної функції, може бути використана як будівельний блок для іншого елемента. Це дозволяє програмі завантажуватися, оскільки розробники можуть створювати нові можливості без написання коду з нуля;

− стійкість. Незалежність сервісу підвищує стійкість застосунків до збоїв. У монолітній архітектурі, якщо один компонент вийде з ладу, це може призвести до збою всієї програми. За допомогою мікросервісів програми

справляються з повним збоєм служби, погіршуючи функціональність і не руйнуючи всі програми.

1.2 Фреймворк React

React − це бібліотека для розробки інтерфейсу користувача на основі JavaScript. Facebook та спільнота розробників з відкритим вихідним кодом керують ним. Хоча React – це бібліотека, а не мова, вона широко використовується у веброзробці. Бібліотека вперше з'явилася в травні 2013 року і в даний час є однією з клієнтських бібліотек, що найчастіше використовуються для веброзробки.

React пропонує різні розширення для всієї архітектурної підтримки застосунків, таких як Flux і React Native, крім простого інтерфейсу користувача.

Популярність React сьогодні перевершила популярність всіх інших фреймворків фронтенд−розробки. Ось чому:

− просте створення динамічних програм. React спрощує створення динамічних вебпрограм, тому що він вимагає меншого кодування і пропонує більше функціональності, на відміну від JavaScript, де кодування часто дуже швидко ускладнюється;

− покращена продуктивність. React використовує віртуальний DOM, тим самим швидше створюючи вебпрограми. Virtual DOM порівнює попередні стани компонентів і оновлює тільки елементи в Real DOM, які були змінені замість того, щоб знову оновлювати всі компоненти, як це роблять звичайні вебзастосунки;

− багаторазові компоненти. Компоненти є будівельними блоками будь−якої програми React, і одна програма зазвичай складається з декількох компонентів. Ці компоненти мають свою логіку та елементи управління, і їх

можна повторно використовувати в усьому застосунку, що, своєю чергою, значно скорочує час розробки програми;

− однонаправлений потік даних. React слідує за односпрямованим потоком даних. Це означає, що при розробці React розробники часто вводять дочірні компоненти в батьківські компоненти. Оскільки дані течуть в одному напрямку, стає легше налагоджувати помилки та знати, де виникає проблема у застосунку на даний момент;

− його можна використовувати для розробки як вебзастосунків, так і мобільних застосунків. Вже відомо, що React використовується для розробки вебзастосунків, але це ще не все, що він може зробити. Існує фреймворк під назвою React Native, похідний від самого React, який є надзвичайно популярним і використовується для створення красивих мобільних застосунків. Таким чином, насправді React можна використовувати для створення вебзастосунків, так і мобільних застосунків;

− спеціальні інструменти для легкого налагодження. Facebook випустив розширення Chrome, яке можна використовувати для налагодження програм React. Це робить процес налагодження вебзастосунків React швидше та простіше.

Вищезгадані причини більш ніж виправдовують популярність бібліотеки React і те, чому вона використовується великою кількістю організацій та підприємств. Тепер пройдемо по функціям React.

React пропонує деякі визначні функції, які роблять його найбільш поширеною бібліотекою для розробки фронтенд−застосунку. Ось перелік цих основних особливостей.

JSX − це синтаксичне розширення JavaScript. Це термін, який використовується в React для опису того, як повинен виглядати інтерфейс користувача. Можна написати HTML структури в тому ж файлі, що і код JavaScript, але його головним правилом є те, що будь−який елемент має мати одного батька. Для цього використовують JSX (або TSX, якщо використовуєте Typescript) (рис. 1.10) [3].

Рисунок 1.10 − JSX формат

Virtual DOM – це полегшена версія Real DOM від React. Реальні маніпуляції DOM значно повільніші, ніж віртуальні маніпуляції DOM. Коли стан об’єкта змінюється, віртуальний DOM оновлює цей об’єкт у реальному DOM, а не всі з них (рис. 1.11) [3].

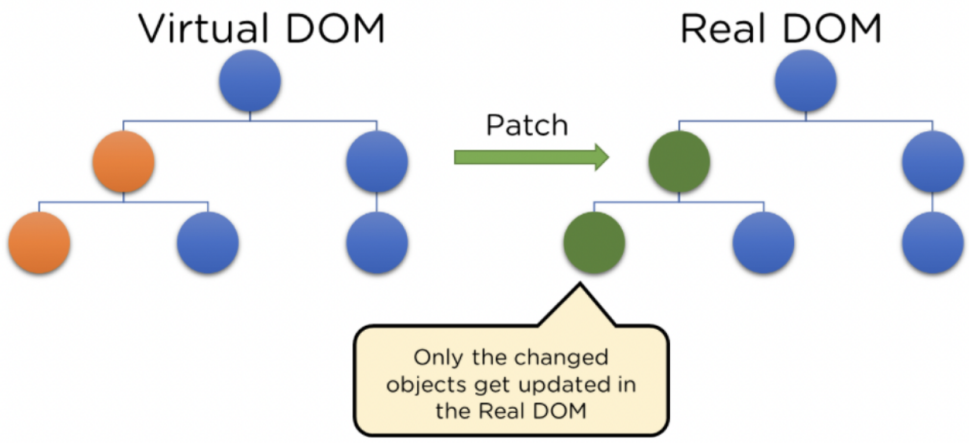

Рисунок 1.11 − Візуалізація Virtual DOM

DOM (Об’єкт−модель документа) розглядає XML або HTML−документ як деревоподібну структуру, в якій кожен вузол є об’єктом, що є частиною документа (рис. 1.12) [3].

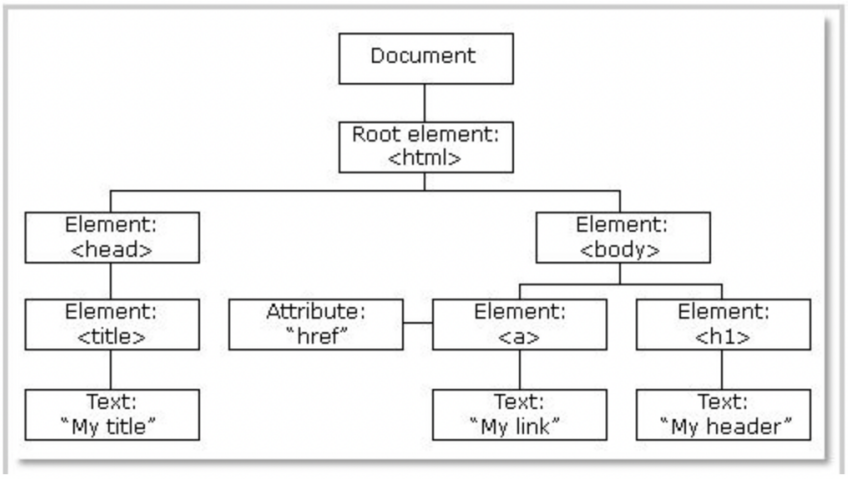

Рисунок 1.12 − DOM структура сторінки

В архітектурі Model View Controller (MVC) React є «Переглядом», відповідальним за зовнішній вигляд та відчуття програми.

MVC − це архітектурний шаблон, який розділяє прикладний шар на модель, вид та контролер. Модель відноситься до всієї логіки, пов’язаної з даними; уявлення використовується для логіки інтерфейсу користувача, а контролер є інтерфейсом між моделлю і уявленням.

React виходить за рамки простого створення фреймворку інтерфейсу користувача; він містить безліч розширень, які охоплюють всю архітектуру програми. Це допомагає створювати мобільні програми та забезпечує рендеринг на стороні сервера. Flux та Redux, серед іншого, можуть розширити React

1.3 Постановка задачі

Об’єктом роботи є створення вебзастосунку для агрегації рекламних інтеграцій.

Метою роботи є розроблення вебзастосунку за допомогою якого можна буде просувати різні компанії на різних платформах.

Для досягнення мети потрібно вирішити такі проблеми:

- навігація між сторінками у вебзастосунку;
- виконати аналіз предметної області;
- відображення великої кількості даних;
- авторизація користувача;
- визначити структуру проєкту;
- виконати моделювання та проєктування вебзастосунку;
- визначити методології, що будуть використовуватись;
- визначити технології та бібіліотеки для розробки вебзастосунку.

2 ОГЛЯД МАТЕМАТИЧНИХ МЕТОДІВ РЕАЛІЗАЦІЇ ЗАСТОСУНКУ

2.1 Docker

Docker – це програмна платформа для швидкої розробки, тестування та розгортання програм. Docker упаковує програмне забезпечення в стандартизовані блоки, які називаються контейнерами. Кожен контейнер включає все необхідне роботи програми: бібліотеки, системні інструменти, код і середовище виконання. Завдяки Docker можна швидко розгортати та масштабувати програми в будь−якому середовищі та зберігати впевненість у тому, що код працюватиме. В основі роботи Docker є стандартизований спосіб виконання коду.

Docker – це операційна система контейнерів. Подібно до того, як віртуальна машина створює віртуальне уявлення апаратного забезпечення сервера (тобто усуває необхідність безпосередньо керувати таким), контейнери створюють віртуальне уявлення серверної операційної системи. Після встановлення на кожен сервер Docker надає доступ до простих команд, необхідних для збирання, запуску або зупинки контейнерів.

Такі послуги AWS, як AWS Fargate, Amazon ECS, Amazon EKS і AWS Batch, спрощують роботу з контейнерами Docker, а також управління ними в будь−якому масштабі.

Docker надає можливість пакувати та запускати програму в слабо ізольованому середовищі, яке називається контейнером. Ізоляція та безпека дозволяють запускати багато контейнерів одночасно на даному хості. Контейнери легкі та містять все необхідне для запуску програми, тому не потрібно покладатися на те, що в даний момент встановлено на хості.

Docker використовує архітектуру клієнт−сервер, як це показано на рисунку 2.1 [4].

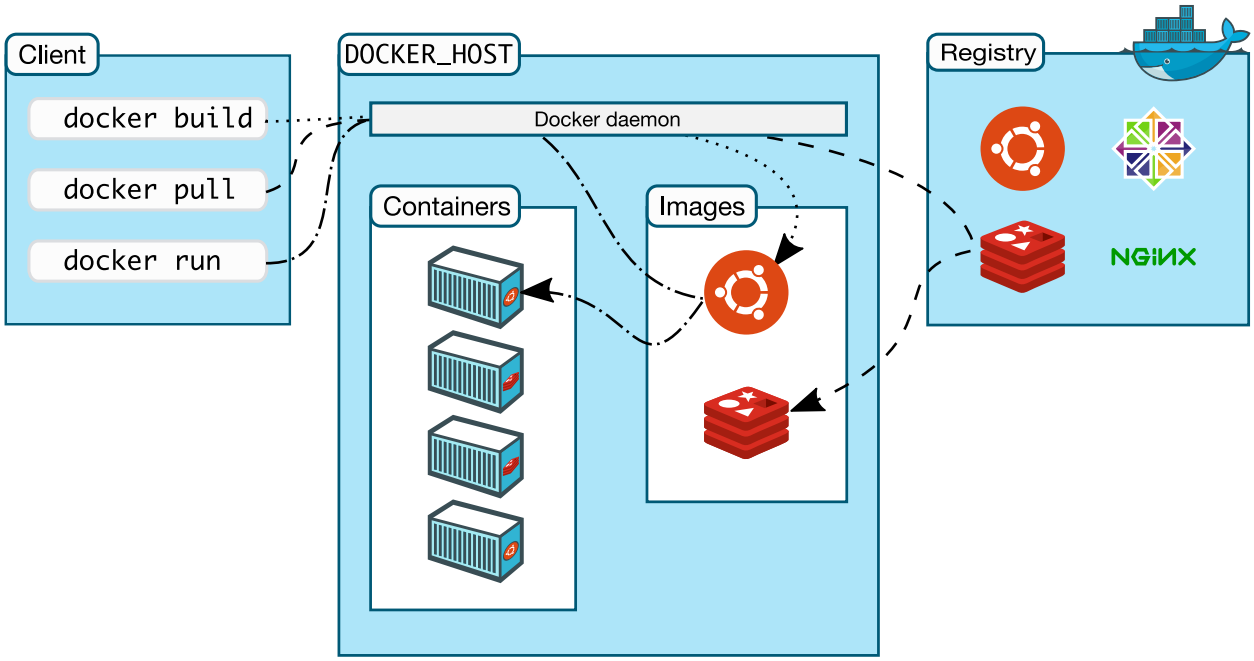

Рисунок 2.1 – Архітектура Docker

Клієнт Docker спілкується з демоном Docker, який займається створенням, запуском та розповсюдженням контейнерів Docker. Клієнт і демон Docker можуть працювати в одній системі, або можно підключити клієнта Docker до віддаленого демона Docker. Клієнт Docker і демон спілкуються за допомогою REST API, через сокети UNIX або мережевий інтерфейс. Іншим клієнтом Docker є Docker Compose, який дозволяє працювати з застосунками, що складаються з набору контейнерів.

Docker написаний мовою програмування Go і використовує переваги кількох функцій ядра Linux для забезпечення його функціональності. Docker використовує технологію, яка називається просторами імен, щоб забезпечити ізольовану робочу область під назвою контейнер. Коли запускається контейнер, Docker створює набір просторів імен для цього контейнера.

Ці простори імен забезпечують рівень ізоляції. Кожен аспект контейнера працює в окремому просторі імен, і його доступ обмежено цим простором імен.

2.2 Атомарний дизайн

Атомарний дизайн – методологія поділу будь−якого інтерфейсу на функціональні компоненти, мінімальні складові одиниці, кожна зі своєю конкретною функцією. Саме ці одиниці потім збиратимуться у функціонуючі системи, що особливо зручно при створенні такого великого документа як дизайн−система. Атомарний дизайн як частина розробки дизайн−системи дозволяє розібрати інтерфейс на мінімальні складові одиниці, які потім збираються у велике ціле. Наприклад, одним із елементів атомарного дизайну є сторінка (вищий рівень конкретики), що складається з набору готових UI−елементів, з яких будуються будь−які вебсайти, програми, ресурси.

Атомарний дизайн представляє методологію створення масштабованих систем, багаторазових компонентів, а також систем проєктування. На перших порах Інтернету користувачі мали обмежену кількість «сторінок», які не були адаптивними або масштабованими. На рисунку 2.2 показано п’ять різних рівнів в атомному дизайні [5].

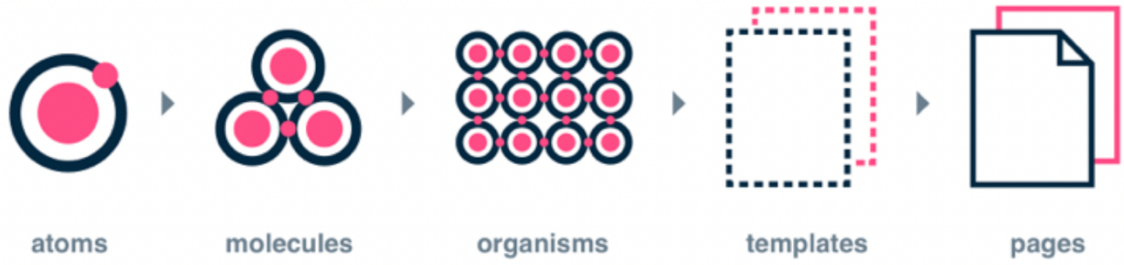

Рисунок 2.2 – Рівні дизайну в Atomic Design

Атоми − мінімальні базові елементи інтерфейсу: кнопки, значки, іконки, вебформи, поля введення. Ці елементи не можна розділити більш дрібні без втрати функціональності.

Молекули – прості компоненти user interface, зібрані з атомів: − Форма пошуку, створена з атомів; кнопки, рядки введення та поля пошуку. Створення

таких молекульних компонентів спрощує тестування, повторне використання, робить інтерфейс більш цільним.

Організми − незалежні та відносно складні за структурою ділянки інтерфейсу, що складаються з груп атомів/молекул. Наприклад, хедер, що складається з логотипу, форми пошуку, пунктів меню для навігації. Також прикладом організму може бути каталог інтернет−магазину взуття або одягу, складений у вигляді сітки.

Шаблони – макети сторінок з базовою структурою контенту (як виглядатиме сайт після запуску). Він містить всі необхідні компоненти і формує структуру контенту. При цьому дуже важливо враховувати те, як може змінюватися контент, встановити обмеження щодо характеристик: кількість символів у заголовку та тексті, розмір зображень.

Сторінки – шаблони, наповнені цим контентом, готові до використання. Саме сторінки демонструють реальну взаємодію UI з контентом, а також дають розуміння ефективності системи проєктування.

2.3 Авторизація використовуючи JWT токени

JSON Web Token (JWT) – це JSON об’єкт, який визначено у відкритому стандарті RFC 7519. Він вважається одним із безпечних способів передачі інформації між двома учасниками. Для створення необхідно визначити заголовок (header) із загальною інформацією по токену, корисні дані (payload), такі як id користувача, його роль тощо. та підписи (signature).

Припустимо, що хочемо зареєструватися на сайті. Приклад показано на рисунку 2.3. У цьому випадку є три учасники − користувач user, сервер application server і сервер автентифікації authentication server. Сервер автентифікації забезпечуватиме користувача токеном, за допомогою якого він пізніше зможе взаємодіяти з програмою [6].

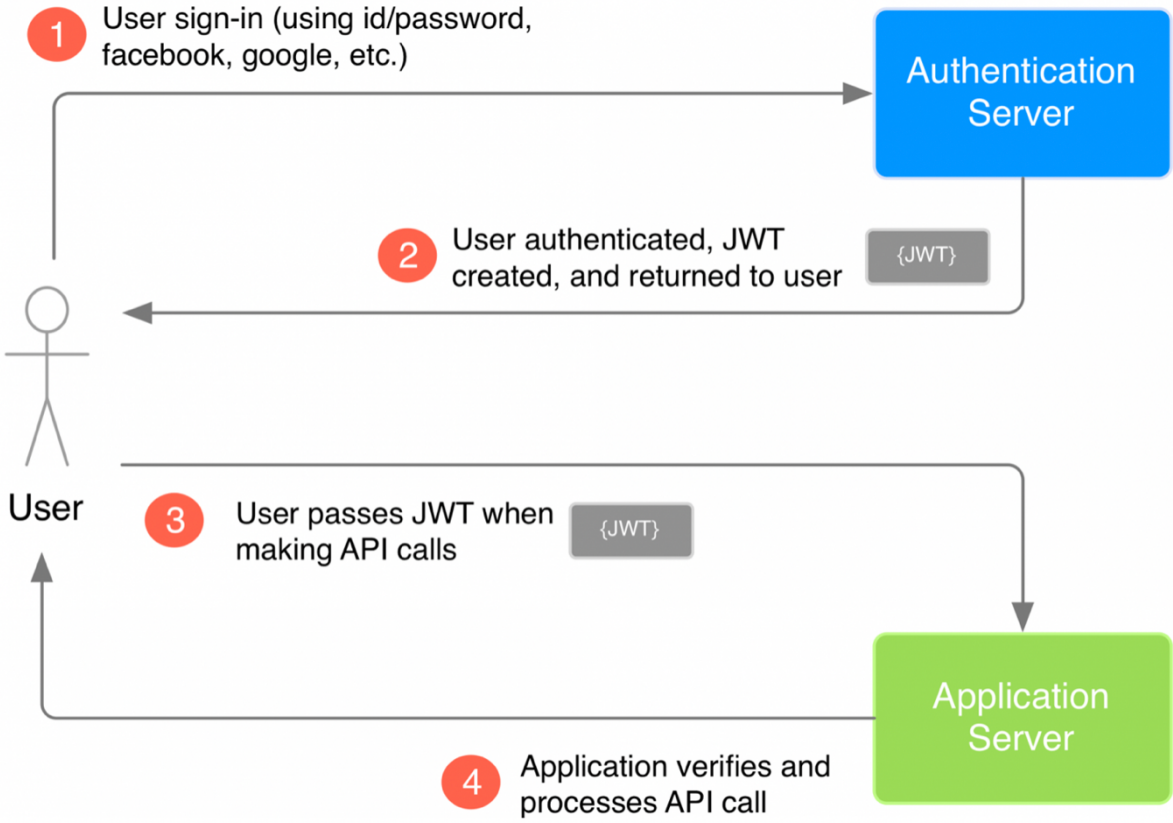

Рисунок 2.3 – Приклад роботи JWT

Застосунок використовує JWT для перевірки аутентифікації користувача таким чином:

Крок 1. Спочатку користувач заходить на сервер аутентифікації за допомогою аутентифікаційного ключа (це може бути пара логін/пароль, або Facebook ключ, або Google ключ, або ключ від іншого обліку).

Крок 2. Потім сервер аутентифікації створює JWT та відправляє його користувачеві.

Крок 3. Коли користувач робить запит до програми API, він додає до нього отриманий раніше JWT.

Крок 4. Коли користувач робить API запит, програма може перевірити за переданим із запитом JWT чи є користувач тим, за кого себе видає. У цій схемі сервер програми налаштований так, що зможе перевірити, чи є вхідний JWT саме тим, що був створений сервером аутентифікації (процес перевірки буде пояснений пізніше більш детально).

Поширеним способом використання JWT є маркери носія OAuth. У цьому прикладі сервер авторизації створює JWT на запит клієнта і підписує його, щоб він не міг бути змінений будь−якою іншою стороною. Потім клієнт надішле цей JWT зі своїм запитом до REST API. REST API перевірить, що підпис JWT відповідає його корисному навантаженню та заголовку, щоб визначити, що JWT дійсний. Коли REST API перевірив JWT, він може використовувати претензії, щоб задовольнити або відхилити запит клієнта.

Простіше кажучи, можно уявити токен носія JWT як значок особи, щоб потрапити в захищену будівлю. Значок поставляється зі спеціальними дозволами (претензії); тобто він може надавати доступ лише до окремих ділянок будівлі. Сервером авторизації в цій аналогії є стійка реєстрації або емітент значка. А щоб переконатися, що бейдж дійсний, на ньому надруковано логотип компанії, схожий на підпис JWT. Якщо власник значка намагається отримати доступ до зони з обмеженим доступом, дозволи на значку визначають, чи можуть він отримати доступ до зони, подібно до заяв у JWT.

Коротше кажучи, JWT використовуються як безпечний спосіб аутентифікації користувачів і обміну інформацією.

Як правило, приватний ключ або секрет використовується емітентом для підписання JWT. Одержувач JWT перевірить підпис, щоб переконатися, що токен не був змінений після того, як його підписав емітент. Неаутентифікованим джерелам важко вгадати ключ підпису та спробувати змінити претензії в JWT.

Проте не всі алгоритми підписання створені однаковими. Наприклад, деякі алгоритми підпису використовують секретне значення, яке спільно використовують емітент і сторона, яка перевіряє JWT. Інші алгоритми використовують відкритий і закритий ключ. Закритий ключ відомий лише емітенту, тоді як відкритий ключ може бути широко поширений. Для перевірки підпису можна використовувати відкритий ключ, але для створення підпису можна використовувати лише закритий ключ. Це більш безпечно, ніж

загальний секрет, оскільки приватний ключ повинен існувати лише в одному місці.

Через це серверу не потрібно зберігати базу даних з інформацією, необхідною для ідентифікації користувача. Для розробників це чудова новина, тому що сервер, який видає JWT, і сервер, який його перевіряє, не повинні бути однаковими.

2.4 Навігація між сторінками за допомогою React−Router

React Router — це стандартна бібліотека маршрутизації (маршрутизації) в React. Він зберігає інтерфейс застосунків синхронізованим з URL−адресою в браузері. React Router дозволяє маршрутизувати «потік даних» у цьому застосунку легким способом.

React Router в самому відділі має переваги та широко використовується в застосунках React більше з боків Сервер, ніж програми React із сторін Клієнта. Точне React Router зазвичай використовується в прикладі React в середовищі NodeJS Server, він дозволяє визначити динамічні URL−адреси, а також відповідні філософії «Single Page Application» (Одностраничное застосунок) в React. Для розробки застосунків React не потрібно писати багато компонентів, потрібен лише один файл для обслуговування користувачів цього index.html (рис. 2.4) [7].

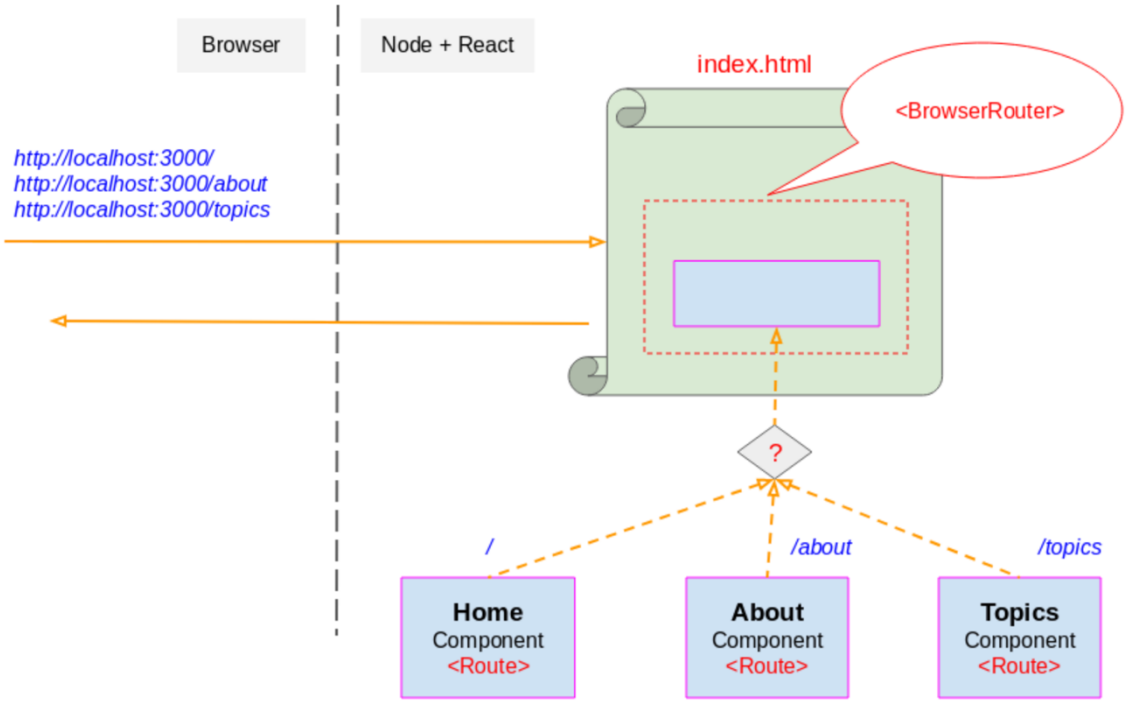

Рисунок 2.4 – Принцип роботи React−Router

React Router допомагає визначити динамічні URL−адреси та вибрати відповідний компонент для візуалізації (зображення) на браузері користувача відповідно до кожного URL.

React Router надає 2 компоненти це <BrowserRouter> і <HashRouter>. Ці два компоненти відрізняються у вигляді URL, які вони створюватимуть і синхронізувати. На рисунку 2.5 представлено як само виглядатиме запроси [7].

```
// <BrowserRouter>
http://example.com/about

// <HashRouter>
http://example.com/#/about
```

Рисунок 2.5 – Приклади компонентів

<BrowserRouter> більш широко використовується, він використовує History API, що має назву в HTML5 для моніторингу історії маршрутизатора. При цьому <HashRouter> використовує хеш в URL (window.location.hash), щоб запам’ятати всі речі. Якщо цікаво підтримувати старі браузери або застосунок React за допомогою маршрутизатора з боку клієнта, то <HashRouter> є правильним вибором.

2.5 Відображення великого обсягу даних за допомогою React−Table

React Table — це набір гачків для створення потужних таблиць і сіток даних. Ці гачки легкі, складні та надзвичайно розширювані, але не відображають розмітку чи стилі для вас. Це означає, що React Table — це бібліотека інтерфейсу користувача без голови [8].

React Table — це безголовна утиліта, що означає, що вона не відтворює і не надає жодних фактичних елементів інтерфейсу користувача. Користувач відповідаєте за використання стану та зворотних викликів хуків, наданих цією бібліотекою, для відтворення власної розмітки таблиці.

Ось коротке пояснення, чому безголовий інтерфейс є важливим:

− розділення проблем. React Table як бібліотека, чесно кажучи, не має жодної відповідальності за інтерфейс користувача. Зовнішній вигляд, відчуття та загальний досвід столу – це те, що робить програму чи продукт чудовими.

− технічне обслуговування. Видаляючи величезну (і, здавалося б, нескінченну) поверхню API, необхідну для підтримки кожного випадку використання інтерфейсу, React Table може залишатися маленькою, легкою у використанні та простою в оновленні/обслуговуванні.

− розширюваність. інтерфейс користувача представляє незліченну кількість крайніх випадків для бібліотеки просто тому, що це творчий засіб, де кожен розробник робить речі по−різному. Не диктуючи проблеми з інтерфейсом, React Table дає розробнику можливість проєктувати та

розширювати інтерфейс користувача на основі їх унікального випадку використання.

2.6 Yarn Workspases

Yarn Workspases — це новий спосіб налаштувати архітектуру пакетів, яка доступна за замовчуванням, починаючи з Yarn 1.0. Це дозволяє налаштувати кілька пакетів таким чином, що потрібно лише один раз запустити yarn install, щоб встановити їх усі за один прохід [9].

Хоча Yarn автоматично вибирає роздільну здатність робочого простору, коли вони збігаються, є випадки, коли абсолютно немає бажання ризикувати використовувати пакет із віддаленого реєстру, навіть якщо версії не збігаються (наприклад, якщо проект насправді не призначений для публікації, і просто треба використовувати робочі області для кращого розподілу коду).

Для цих випадків Yarn тепер підтримує новий протокол розділення, починаючи з v2:workspace:. Коли використовується цей протокол, Yarn відмовиться розв’язувати що-небудь інше, крім локальної робочої області.

Коли робоча область запакована в архів (незалежно від того, через yarn pack або через одну з команд публікації, як-от yarn npm publish), динамічно замінюємо будь-яку залежність workspace на:

− відповідна версія в цільовому робочому просторі (якщо використовується *, ^, ~ або шлях відносно проекту);

− пов'язаний діапазон semver (для будь-якого іншого типу діапазону).

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

3.1 Етапи створення вебзастосунку

3.1.1 Створення скелету вебзастосунку

На цьому етапі перш за все створюється скелет вебзастосунку. Оскільки цей проєкт може буде великий, то потрібна найбільш зрозуміла структура яку можна побачити на рисунку 3.1

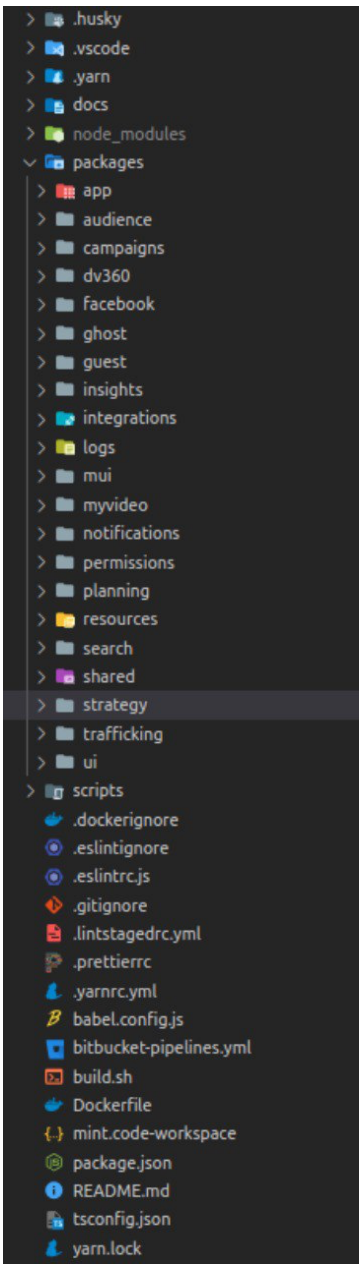

Рисунок 3.1 – Структура вебзастосунку

Пояснення щодо структури:

– .husky. Це pre-commit(запускається спочатку, перш ніж навіть введено повідомлення про фіксацію. Він використовується для перевірки моментального знімка, який має бути зафіксований, щоб перевірити, чи було щось забуто, щоб переконатися, що тести запущені, або щоб перевірити все, що потрібно перевірити в коді) хукі який забезпечує якість назв комітів та перевірку якості кода;

– node_modules. Місце в якій зберігаються всі бібліотеки, модулі та залежності;

– packeges. Кореневий модуль для об'єднання всіх мікросервісів;

– app. Модуль для з'єднання сервісів через React–Router;

– audience. Модуль для забезпечення ауедієнції між компаніями;

– campaigns. У сервісі компанії можна отримати інформацію про кожну компанію клієнта, дізнатися виділений на компанію бюджет, створити нову компанію, з цієї сторінки можна перейти на платформи інтеграції;

– dv360. Кореневий модуль для надання послуг легкого користування з платформою;

– gost. Сторінка налаштувань інтеграції з якої можна лише отримувати певні дані, але не можна туди відправляти;

– guest. Сторінка логіна користувача, на якій він може зайти в програму або зареєструватися. Якщо користувач введе невірний логін або пароль, він обов'язково отримає повідомлення про помилку;

– insights. На цій сторінці можна отримувати більш поглиблену аналікіку;

– integrations. На сторінці інтергацій користувач може підключати як інтеграції віртуального типу, так і двосторонні інтергарції. Такими інтеграціями двостороннього типу може бути фейсбук, гугл і т.д;

– mui. Бібліотека з компонентами для забезпечення конценсу між сервісами програми, щоб у кожній сторінці були однакові компоненти;

– resourses. Сервіс який є утилітою для інших сервісів та надає ресурси у вигляді картинок та svg які відображаються на інших сторінках;

– planning. Сторінка для налаштування в компанії всіх каналів платформ і налаштувань кожної з платформ, є таблицею, що використовує React-Table;

– search. Надає настройки для каналів типу search, одні з таких інтеграцій, які можуть знаходитися там, це Google Display Network. Забезпечує взаємодію з інтеграцією Goggle Ads;

– shared. Пакет у якому знаходяться функції та різного виду утиліти та константи які можуть використовуватись в інших мікросервісах;

– ui. Пакет для утилітів який у собі містить утиліти та абстрактні компоненти у вигляді кнопок і дропдаунів, які також містяться в інших компонентах, так само містить у собі компонент який можна змінювати за допомогою впровадження залежності. Цей підхід дозволяє полегшити розробку оптимізації застосунку;

– eslintrc.js. Файл який містить правила для опису стилю коду;

– gitignore. Забезпечує взаємозв’язок pre-commit хукамі для гіта;

– prettierrc. Забезпечує форматування кода за заданими правилами eslintrc.js;

– yarnrc.yml. Описує метадані node_modules;

– babel.config.js. Конфігурація яка налаштовує поліфіти для перетворення коду з ES 6 стандарту на ES 5 і нижче;

– Dockerfile. Забезпечує конфігурації контейнерізації програми, містить у собі зовнішні залежності та пакети для роботи програми.

3.1.2 Реалізація кореневого модуля вебзастосунку

На рисунку 3.2 показано реалізація кореневого модуля вебзастосунку, що включає в себе сайдбар по якому можна робити навігацію по всьому

вебзастосунку, який ділиться на декілька мікросервісів, що вставляються по відповідному роутингу.

```
<UserPermissionsContext.Provider
  value={{ permissions: profileData!.profile.permissions }}
>
  <SidebarWrapper profileData={profileData}>
    <Switch>
      {allowedRoutes.map((route) => (
        <Route
          key={Array.isArray(route.path) ? route.path[0] : route.path}
          path={route.path}
          component={route.component}
          exact={!!route.exact}
        />
      ))}
      {defaultRoute ? (
        <Redirect to={defaultRoute} />
      ) : (
        <div>No access</div>
      )}
    </Switch>
  </SidebarWrapper>
  <div id="dialog-portal" />
</UserPermissionsContext.Provider>
```

Рисунок 3.2 – Лістинг коду кореневого модуля

Тепер створивши компоненти сайдбару на головній сторінці, можна подивитися на те, як це буде виглядати зараз на вебзастосунку (рис. 3.3).

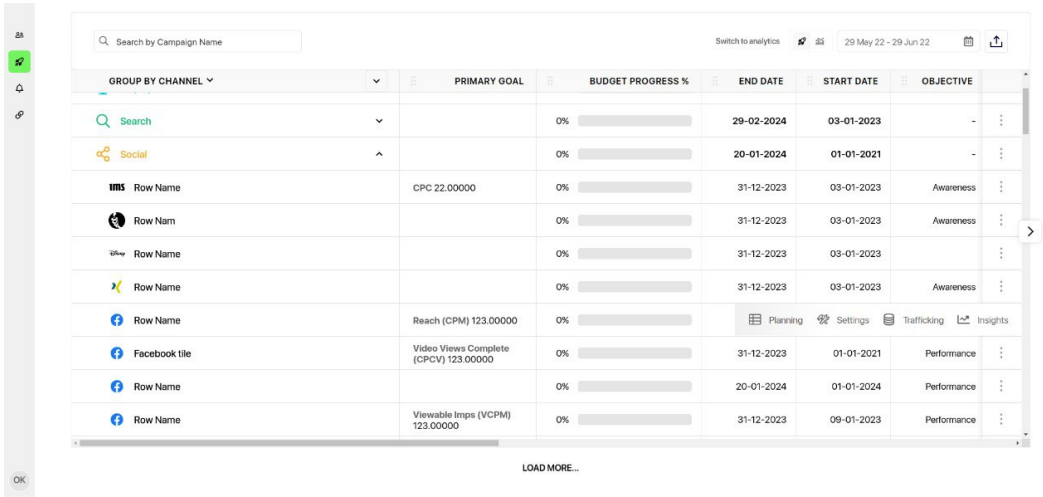

Рисунок 3.3 – Головна сторінка вебзастосунку

3.2 Реалізація авторизації при виконанні запитів

На рисунку 3.4 показано реалізація авторизації при виконанні запитів до серверу та валідація JSON вебтокену.

```
let tokenUpdatingPromise: Promise<AxiosResponse<any>> | null = null;
const applyAccessToken = async (request: AxiosRequestConfig) => {
  if (import.meta.env.MODE === 'test' || auth === null) {
    return request;
  }

  if (tokenUpdatingPromise) {
    await tokenUpdatingPromise;
  }

  if (auth.refreshTokenExpired()) {
    auth.logout();
    throw new Error('Refresh token expired');
  }

  if (auth.accessTokenExpired()) {
    tokenUpdatingPromise = auth.updateTokens();

    try {
      await tokenUpdatingPromise;
    } catch (error) {
      auth.logout();
      throw new Error('Failed update tokens');
    } finally {
      tokenUpdatingPromise = null;
    }
  }

  return R.assocPath(
    ['headers', 'Authorization'],
    `jwt ${auth.access}`,
    request,
  );
};
```

Рисунок 3.4 – Лістинг коду для реалізація авторизації при виконанні запитів

3.3 Реалізація інтерфейсу компаній

На цьому етапі показано реалізацію інтерфейсу компанії, що за певний проміжок хоче отримати прибуток від своїх інтеграцій на певних інтеграційних платформах (рис.3.5).

```
export type BuilderTableRow = Partial<{
  id: number;
  idx?: number;
  depth: number;
  cells: Array<any>;
  getRowProps: (props?: { style: CSSProperties }) => any;
  index?: string;
  original: {
    order?: number;
    channel?: string;
    platformMedia?: string;
    budget?: string;
    currency?: {
      sign: string;
    };
    currencySign: string;
    id: number;
    type: string;
    campaign?: number;
    errors: {
      [name: string]: {
        error: string;
        value: string;
      };
    };
    icon?: string;
    parent: number;
    subRows: Array<{ id: number }>;
    parentInfo: { name: string; dateFrom?: string; dateTo?: string };
    dateFrom?: string;
    dateTo?: string;
    nameKey: string;
    isConfigured?: boolean;
    label?: string;
    isCopy?: boolean;
    canBeDeleted?: boolean;
    name?: string;
  };
  canExpand: boolean;
  isExpanded: boolean;
  getToggleRowExpandedProps: (props?: { title: string | null }) => any;
  toggleRowExpanded: () => void;
}>;
```

Рисунок 3.5 – Лістинг коду для реалізація інтерфейсу компаній

3.4 Алгоритм з’єднання бізнес логік вебзастосунку

На рисунку 3.6 показано з’єднання усіх редьюсерів.(редьюсор - це чиста функція яка приймає попередній стейт та повертає новий, вона не повинна мати ніяких сторонніх эффектів, що можуть повести за собою неочікувану повединку вебзастосунку. За допомогою функції комбайн редьюсорс функції редьюсоры з’єднюються в один. Це допомогає зберегати стан застосунку централізовано та мати доступ до стану в будь якому його місці).

```
Alex Hanga, 3 months ago | 2 authors (Maxym Lozovoy and others)
1  import { combineReducers, StateFromReducersMapObject } from 'redux';
2
3  import adobeAnalytics from './adobeAnalytics';
4  import campaigns from './campaigns';
5  import googleAnalytics from './googleAnalytics';
6
7  const reducers = {
8    campaigns,
9    googleAnalytics,
10   adobeAnalytics,
11 };
12 const rootReducer = combineReducers(reducers);
13
14 export type RootState = StateFromReducersMapObject<typeof reducers>;
15
16 export default rootReducer;
17
```

Рисунок 3.6 – Лістинг коду для з’єднання редьюсорів

3.5 Реалізація серверної частини до двухсторонньої інтеграції

На рисунку 3.7 показано реалізація серверної частини до двухсторонньої інтеграції за допомогою проміжного програмного забеспечення що

називається Redux-Saga, де використовується функція takeLatest, що поліпшує продуктивність вебзастосунку.

```
export default function* adobeAnalyticsSaga() {
  yield takeEvery(
    AdobeAnalyticsActions.loadAdobeIntegration.request.type,
    loadIntegration,
  );
  yield takeLatest(
    AdobeAnalyticsActions.loadAdobeAnalyticsReports.request.type,
    loadReports,
  );
}
```

Рисунок 3.7 – Лістинг коду реалізації серверної частини до двухсторонньої інтеграції

ВИСНОВКИ

У рамках кваліфікаційної роботи було розроблено вебзастосунок для агрегації рекламних інтеграцій

У ході роботи було створено навігацію між сторінками у вебзастосунку за допомогою бібліотеки React-Router, що надає можливість переключатися між сторінками в одностраничному вебзастосунку. Виконано аналіз предметної області комппанії, що займається рекламною агрегацією, було створено технічне завдання по якому був розроблений вебзастосунок. Також була реалізована можливість відображати велику кількість даних за допомогою бібліотеки React-Table та підходу віртуалізації. Створена авторизація користувача за допомогою підходу використання JWT токену, що містить у собі access та refresh токен, що зберігаються в локальному сховищі браузера та допомогають авторизувати користувача при виконанні запитів на сервер. Визначено структуру проєкту з використанням мікросервісної архітектури та атомарного дизайну. Виконано моделювання та проєктування вебзастосунку за допомогою патернів проєктування систем. Інформаційна система має достатню кількість функцій для використання та експлуатації в бізнес сфері. Описано технології за допомогою яких вебзастосунок був створений.

У результаті роботи здійснена програмна реалізація готового вебзастосунку для споживача.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Masse, M. (2011). REST API design rulebook: designing consistent RESTful web service interfaces. " O'Reilly Media, Inc." (pp. 213-226).

2. Загальні відомості про мікросервісну архітектуру. URL: https://microservices.io (дата звернення 22.04.2022).

3. Загальні відомості про фрейморк React. URL: https://www.w3schools.com/whatis/whatis_react.asp (дата звернення 31.04.2022).

4. Документація Docker. URL: https://www.docker.com (дата звернення 29.04.2022).

5. Atomic Design Methodology.

URL: https://atomicdesign.bradfrost.com/chapter-2/ (дата звернення 03.05.2022).

6. Загальні відомості про JSON Web Token. URL: https://auth0.com/docs/secure/tokens/json-web-tokens (дата звернення 01.05.2022).

7. React Router v6. URL: https://reactrouter.com (дата звернення 16.04.2022).

8. Загальні відомості про React Table. URL: https://mui.com/material-ui/react-table/ (дата звернення 14.04.2022).

9. What A Yarn Workspace Is, And The Problem It Solves. URL: https://planflow.dev/blog/what-a-yarn-workspace-is-and-the-problem-it-solves (дата звернення 24.04.2022).

10. What is SOAP: Formats, Protocols, Message Structure URL: https://www.altexsoft.com/blog/engineering/what-is-soap-formats-protocols-message-structure-and-how-soap-is-different-from-rest/ (дата звернення 03.05.2022).

11. Gackenheimer, C., & Paul, A. (2015). Introduction to React (Vol. 52). Apress (pp. 137-150).

12. Boduch, A. (2017). React and React Native. Packt Publishing Ltd.

13. Banks, A., & Porcello, E. (2017). Learning React: functional web development with React and Redux. " O'Reilly Media, Inc." (pp. 1-15).

14. Fedosejev, A. (2015). React. js essentials. Packt Publishing (pp. 557-655).

15. Guha, A., Saftoiu, C., & Krishnamurthi, S. (2010, June). The essence of JavaScript. In European conference on Object-oriented programming (pp. 126-150). Springer, Berlin, Heidelberg (pp. 17-90).

16. Rastogi, A., Swamy, N., Fournet, C., Bierman, G., & Vekris, P. (2015, January). Safe & efficient gradual typing for TypeScript. In Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (pp. 167-180).

17. Galloway, J., Haack, P., Wilson, B., & Allen, K. S. (2012). Professional ASP. NET MVC 4. John Wiley & Sons (pp. 1-102).

18. Rai, R. (2013). Socket. IO Real-time Web Application Development. Packt Publishing Ltd (pp. 167-180).

19. Duckett, J. (2011). HTML & CSS: design and build websites (Vol. 15). Indianapolis, IN: Wiley (pp. 43-90).

20. Boduch, A. (2019). React Material-UI Cookbook: Build captivating user experiences using React and Material-UI. Packt Publishing Ltd (pp. 16-51).

21. Cantelon, M., Harter, M., Holowaychuk, T. J., & Rajlich, N. (2014). Node. js in Action (pp. 17-20). Greenwich: Manning.

22. Cadenhead, T. (2015). Socket. IO Cookbook. Packt Publishing Ltd.

23. Satheesh, M., D'mello, B. J., & Krol, J. (2015). Web development with MongoDB and NodeJs. Packt Publishing Ltd.

24. Petrou, M. M., & Petrou, C. (2010). Image processing: the fundamentals. John Wiley & Sons.

25. Boyd, R. (2012). Getting started with OAuth 2.0. " O'Reilly Media, Inc.".

26. Necula, G. C. (1998). Compiling with proofs. Carnegie Mellon University.

27. Abbott, M. L., & Fisher, M. T. (2015). The art of scalability: Scalable web architecture, processes, and organizations for the modern enterprise. Addison-Wesley Professional.

28. Pham, A. D. (2020). Developing back-end of a web application with NestJS framework: Case: Integrify Oy's student management system.

29. Mäntylä, M. (1987). An introduction to solid modeling. Computer Science Press, Inc..

30. Vernon, V. (2013). Implementing domain-driven design. Addison-Wesley.