Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет електроніки та комп’ютерних технологій

Кафедра радіоелектронних
і комп’ютерних систем

КУРСОВА РОБОТА
на тему: **“РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ SSP ПЛАТФОРМИ”**

Виконав ст. 5 курсу
гр. ФЕІм-14
_____Рішко В. М.
(дата)
Науковий керівник
_____ас. Футей О. В.
(дата)

Львів 2023

ЗМІСТ

ВСТУП……...3
1. ПРОГРАМАТИЧНА РЕКЛАМА……...5
1.1. Огляд програматичної реклами та основних понять…….5
1.2. Протокол OpenRTB…….11
1.3. SSP платформи……17
2. РОЗРОБКА ПРОЕКТУ.....22
2.1. Структура проекту.....22
2.2. Інтерфейс додатку.....26
ВИСНОВКИ.....29
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....30

ВСТУП

Розробка і впровадження рекламних стратегій та кампаній в онлайн-середовищі стають все складнішими і конкурентнішими завдяки стрімкому розвитку цифрових технологій. В такому швидкозмінному середовищі, де кількість платформ, додатків та медіа-контенту невпинно зростає, ефективне керування рекламою стає ключовим фактором успіху для компаній.

Одним з найбільш важливих інструментів для оптимізації рекламних процесів є торг в режимі реального часу (Real-Time Bidding, RTB). RTB дозволяє автоматизувати процеси купівлі та продажу реклами, використовуючи алгоритми та програмні платформи для миттєвого визначення ціни та розміщення рекламних пропозицій.

Один з ключових інструментів, що допомагає автоматизувати продаж рекламного простору, є SSP (supply side platform). SSP дозволяють видавцям з'єднатися з різними мережами рекламних агентств, обмінниками реклами та платформами DSP, щоб продавати свій рекламний інвентар на основі показів. Завдяки SSP, видавці реклами отримують зручний та потужний інструмент для управління своїм рекламним інвентарем, а рекламодавці матимуть можливість швидко досягати цільової аудиторії та ефективно розміщувати свої рекламні пропозиції.

Одним з ключових етапів у розробці клієнтської частини SSP платформи є аналіз вимог та потреб видавців. Розуміння їхніх вимог та бізнес-процесів дозволить створити функціонал, що відповідає їхнім потребам. Наприклад, це можуть бути можливості управління рекламним інвентарем, налаштування фільтрів для підбору релевантних рекламних пропозицій, аналітика та звітність про ефективність рекламних кампаній та інші функції, що сприяють оптимізації рекламного процесу.

Для розробки клієнтської частини SSP платформи використовуються сучасні технології та методики програмування. Застосування відкритих стандартів та протоколів дозволяє забезпечити сумісність та інтеграцію з іншими

рекламними системами та платформами. Крім того, важливим елементом розробки є забезпечення безпеки та захисту персональних даних користувачів, дотримання відповідних регуляторних вимог та політик конфіденційності.

Тема цієї роботи полягає у розробці клієнтської частини SSP платформи. Основною метою розробки є створення потужного та ефективного інструменту для видавців реклами, який дозволить їм автоматизувати процес продажу свого рекламного інвентару. Ця робота спрямована на розгляд основних принципів програматичної реклами та ключових аспектів розробки платформи та дослідження її можливостей та переваг.

1. ПРОГРАМАТИЧНА РЕКЛАМА

1.1. Огляд програматичної реклами та основних понять.

Програматична реклама є автоматизованою формою закупівлі рекламного простору в цифровому просторі. Традиційний процес ручної закупівлі медіа включав налаштування кампаній у рекламному сервері рекламодавця, після чого рекламні коди надсилалися видавцю (наприклад, за допомогою електронної пошти). Команда Ad Ops на стороні видавця розміщувала код в рекламному сервері, після чого починалася кампанія.

Існує багато переваг включення програматичної реклами в маркетингову стратегію.

Деякі з основних переваг:

- Великомасштабне охоплення аудиторії.
- Дані та аналітика в реальному часі.
- Можливість використовувати власні та сторонні дані.
- Можливості для стратегій кампаній на різних пристроях.
- Великомасштабне охоплення аудиторії.
- Можливість росту та масштабування.

Програматична реклама є одним з найефективніших та економічних типів реклами на сьогоднішній день. Середні вартості на тисячу показів (CPM) для програматичної реклами становлять від 0,50 до 2,00 доларів. Навіть з невеликим бюджетом маркетингові витрати можуть принести значний результат у досягненні цільової аудиторії та збільшенні впізнаваності продукту.

Оскільки програматичні платформи ґрунтуються на торгівлі в реальному часі, рекламодавці отримують перевагу в майже миттєвому отриманні даних.

Ще одна перевага програматичної реклами полягає в типах сегментів даних, доступних для рекламодавців. Наприклад, рекламодавці можуть завантажити власні персональні дані в безпечний спосіб і спрямовувати їхню

рекламу безпосередньо на цих людей, використовуючи сигнали торгівлі в реальному часі.

Платформи управління попитом (DSP) мають багато сторонніх сегментів, на які рекламодавці можуть спрямовуватися, якщо вони не мають власних персональних даних. Інший тип сторонніх даних, яким рекламодавці можуть скористатися – це перетворення їхніх персональних даних в сторонні дані шляхом створення аудиторій, подібних до їхніх власних клієнтів. Цей підхід відкриває можливості для знаходження нових клієнтів, які схожі на існуючих.

Real-time bidding (RTB) є підгрупою програматичної реклами, що сприяє купівлі та продажу рекламних інвентарів через аукціон, який відбувається за час, необхідний для завантаження веб-сторінки. RTB відбувається на цифровій біржі, яка дозволяє проведення угод між рекламодавцями та видавцями в режимі реального часу. Цей процес стосується пошуку, дисплейної та відео-реклами комп'ютерів та мобільних пристроїв.

Основною перевагою RTB є те, що він дозволяє рекламодавцям акцентувати увагу на кількості кліків та/або показів, які, ймовірно, отримає реклама від цільової аудиторії. Він дозволяє маркетологам придбати дешевий інвентар на різних веб-сайтах, а не лише на "преміумних" сайтах. Завдяки відстеженню на кількох пристроях, RTB надає рекламодавцям більший контроль над спрямовуванням на осіб, яких вони вважають найціннішими, купуючи рекламний простір за кожен показ, а не оптом за фіксовану ціну. Маркетологи можуть визначити, скільки витрачено на конвертовану цільову аудиторію (наприклад, на тих, хто придбав продукт).

Для видавців RTB дозволяє максимізувати продаж усього доступного рекламного простору за максимальною можливою вартістю, що надає можливість максимізувати прибуток.

Процес ставок в публічному аукціоні або відкритому ринку використовує протокол і специфікацію API OpenRTB, розроблені

Лабораторією технологій Interactive Advertising Bureau (IAB). Ланцюжок постачання RTB включає численних учасників у цифровій рекламній екосистемі.

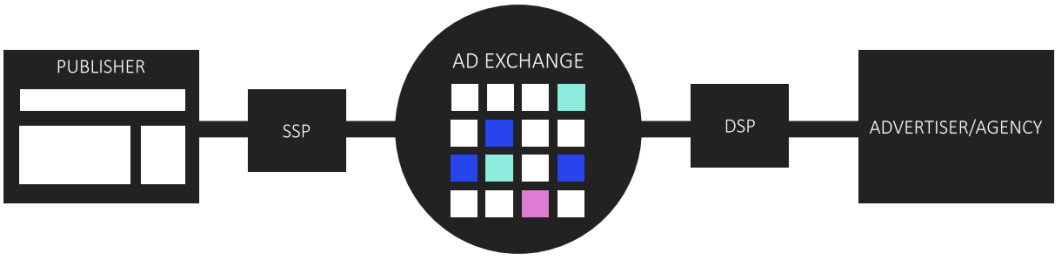

Рис. 1.1. Схема взаємодії між частинами RTB системи.

У контексті цифрової рекламної екосистеми:

- **Publisher** (видавець) – це веб-сайт або додаток, який отримує дохід від показу реклами при відвідуванні користувачем. Простір, який видавці відводять для показу реклами називається інвентарем. За цим широким визначенням, видавцем може бути все – від новинного ресурсу до сторінки блогу, а також включає такі сайти, як Facebook і Google.
- Платформа постачання реклами (**SSP**) допомагає видавцям управляти/продавати свій інвентар на кількох рекламних біржах автоматизованим способом. Вона аналізує інформацію про користувача і надсилає її на біржу для максимізації ціни, яку видавці можуть отримати за свої покази.
- **Рекламна біржа** виступає як онлайн-ринок, який дозволяє рекламодавцям (покупцям) і видавцям (продавцям) купувати та продавати рекламний інвентар. Вона робить це шляхом аукціону.
- Платформа попиту (**DSP**) є еквівалентом постачальної платформи для рекламодавців. Вона дозволяє рекламодавцям зберігати свої оголошення або рекламні матеріали, відстежувати метрики та встановлювати параметри купівлі для своїх кампаній. Тут DSP використовує алгоритми для визначення "вартості" користувача на основі цільової аудиторії, вибраної для рекламної кампанії.

- Платформа управління даними (**DMP**) дозволяє рекламодавцям збирати та управляти користувацькими даними з різних джерел в одному місці. Рекламодавці використовують платформу управління даними для створення аудиторних сегментів, сприяють спрямованості реклами, профілюють користувачів і висновують інтереси на основі даних про користувачів, а також аналізують дані, щоб оцінити успішність поточних рекламних кампаній (DMP не відображена на діаграмі вище).
- **Рекламодавці**, як комерційні, так і некомерційні, створюють рекламу для просування своїх товарів і послуг. Це часто відбувається за допомогою рекламної агенції.

RTB дозволяє купувати та продавати рекламні запаси на основі кожного відображення шляхом проведення торгів у режимі реального часу. Він отримав таку назву, оскільки весь процес відбувається протягом приблизно 100 мілісекунд. Процес виглядає так:

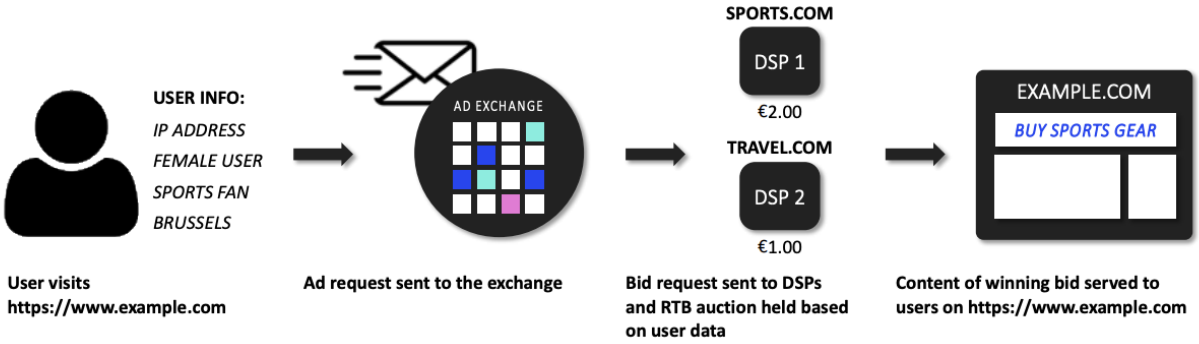

Рис. 1.2. Приклад процесу отримання реклами в RTB системі.

1. Користувач відвідує веб-сторінку. Коли веб-сторінка починає завантажуватися в веб-браузері, це ініціює завантаження рекламного відображення.
2. Відправляється запит до відкритої рекламної біржі через SSP, що вказує на наявність відображення. У цьому запиті вказується URL веб-сайту, а також може містити дані користувача (з використанням cookies) такі як пристрій, розташування стать, вік та іншу корисну інформацію.

3. Відкрита рекламна біржа надсилає запит на торги разом з супровідними даними до кількох учасників (агентств торгівельних столів, які використовують DSP або власну технологію, які збирають ставки від своїх рекламодавців).
4. На основі інформації, що міститься в запиті на торги, DSP автоматично ставить ставки в реальному часі відповідно до даних користувача, цілей маркетолога та заздалегідь встановлених правил бюджету для того, щоб їхнє оголошення було показано користувачу.
5. Переможець з найвищою ставкою отримує рекламне відображення, доставляє оголошення та платить на $0.01 більше, ніж друга найвища ставка – другий ціновий аукціон.
6. Рекламна біржа надсилає URL переможного оголошення та ціну до рекламного сервера видавця, який потім повідомляє веб-браузеру користувача, що показувати.
7. Креатив показується користувачу в браузері.

Важливо відмітити, що RTB включає регулювання збирання, використання та передачі персональних даних користувачів, включаючи вимоги до згоди і конфіденційності, а також для боротьби зі шахрайством, ботами та шкідливими програмами, захист користувачів від небажаної реклами та іншого шкідливого впливу. Прикладами таких решуляцій є COPPA (Закон про захист конфіденційності дітей в інтернеті) в Сполучених Штатах, GDPR в Європейському Союзі, California Consumer Privacy Act (CCPA) в Каліфорнії та інші подібні законодавчі акти, які регулюють збір та використання персональних даних.

Існують й інші типи програматичної реклами, які можуть використовуватися в поєднанні з RTB обмінами або незалежно від них.

Private Marketplace (PMP) є аукціоном "лише за запрошенням", де один або кілька видавців запрошують певних попередньо затверджених покупців зробити ставки на їхні рекламні майданчики. Тут DSP (платформа попиту) безпосередньо підключається до джерела інвентарю видавця, що усуває

потребу у обміні, і покупець точно знає, де буде розміщена реклама. Рекламодавці можуть використовувати приватні ринки, щоб отримати "преміальні" розміщення, поєднуючи їх з торгівлею на відкритому рекламному обміні.

Programmatic Direct — це підхід, який не базується на аукціоні і дозволяє рекламодавцям попередньо купувати гарантовані рекламні враження на певних сайтах видавців. Програматичний директ, мабуть, пропонує перевагу збільшеної прозорості, що є одним із недоліків RTB, і існує два типи програматичного директу. **Programmatic Direct** — це заздалегідь визначена зобов'язаність рекламодавця купувати фіксовану кількість інвентару за фіксованою вартістю за тисячу показів або кліків (вартість за тисячу показів — CPM) на певних сайтах видавців. Видавці можуть бути більш схильними продавати ексклюзивні розміщення, наприклад, рекламу на головній сторінці за фіксовану ціну за гарантовану кількість показів.

Preferred Deal — це заздалегідь визначена домовленість про ціну інвентару між одним покупцем і одним продавцем, але без визначення обсягу інвентару. Цей підхід дозволяє установити фіксовану ціну за інвентар, але не визначає точну кількість доступного обсягу. При цьому покупець має перевагу при розміщенні заявок на інвентар у порівнянні з іншими покупцями, але не має гарантії на отримання всього обсягу.

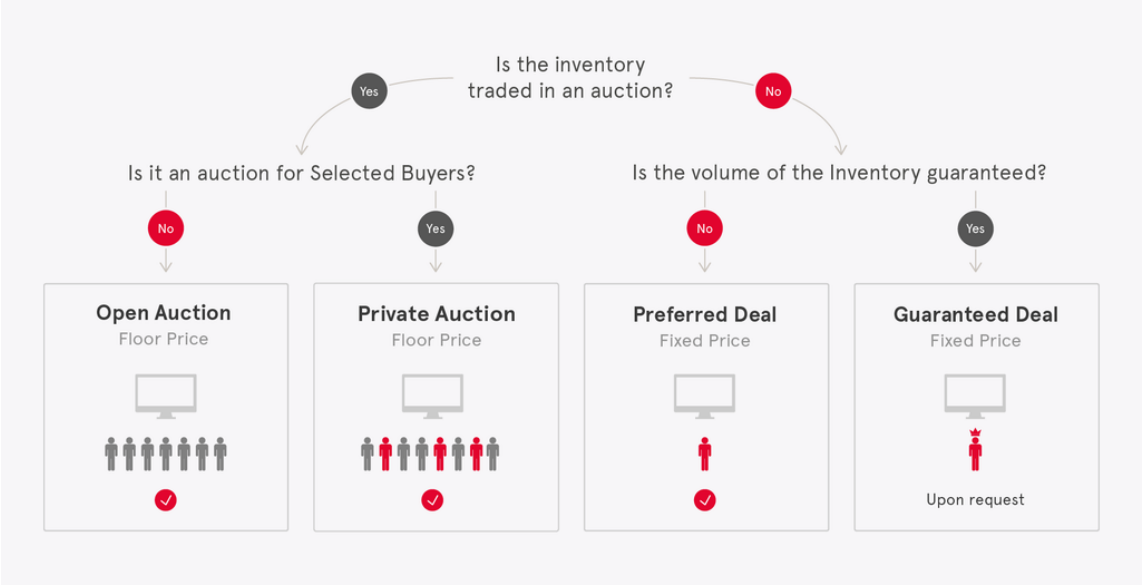

Рис. 1.3. Схема вибору типу програматичної реклами.

Деякі платформи, які відображають рекламу та мають власні рекламні обміни, мають унікальні процеси торгівлі. Наприклад, Facebook використовує Facebook Ads для доставки реклами. Аукціон Facebook Ads працює за принципом "найвищої загальної вартості" і враховує стратегію ставок рекламодавця, оцінку можливості користувача здійснити дію та релевантність реклами конкретному користувачу для визначення переможця аукціону.

1.2. Протокол OpenRTB

OpenRTB (Open Real-Time Bidding) – це відкритий протокол, стандарт, який використовується в програматичній рекламі для автоматизованого купівлі та продажу рекламного інвентару в режимі реального часу (real-time bidding). OpenRTB надає засоби для комунікації між рекламодавцями, торговими платформами (demand-side platforms – DSPs), постачальниками рекламного інвентару (supply-side platforms – SSPs) та іншими учасниками екосистеми програматичної реклами.

Протокол OpenRTB визначає спосіб передачі даних про рекламні запити, пропозиції та іншу інформацію між різними системами у режимі реального часу. Він включає в себе структурований формат даних (JSON або XML), який використовується для обміну інформацією, а також набір полів для передачі деталей про рекламні кампанії, цільову аудиторію, умови торгівлі, ставки та інші параметри.

Використання протоколу OpenRTB дозволяє автоматизувати процес купівлі та продажу рекламного інвентару, забезпечує ефективність, швидкість та прозорість у роботі різних учасників рекламного ринку. Він створює основу для роботи RTB-обмінів, де рекламодавці можуть зробити ставку за рекламний інвентар на основі розуміння цільової аудиторії та інших параметрів, а видавці можуть продавати свій інвентар на аукціонах, де виграє найвища ставка.

OpenRTB було запущено як пілотний проект між трьома платформами DSP (DataXu, MediaMath і Turn) та трьома платформами SSP (Admeld, PubMatic і The Rubicon Project) у листопаді 2010 року. Першою метою було стандартизувати комунікацію між сторонами для обміну списками блокування. Версія 1.0 специфікації OpenRTB для списків блокування була випущена у грудні 2010 року. Після позитивної реакції від галузі, компанія Nexage звернулася до проекту OpenRTB з пропозицією створити специфікацію API для OpenRTB з фокусом на протокол обміну запитами/відповідями в режимі реального часу, зокрема для підтримки мобільної реклами. Було створено мобільний підкомітет між компаніями, що представляють сторону попиту (DataXu, Fiksu і [X+1]) та компаніями, що представляють сторону пропозиції (Nexage, Pubmatic, Smaato і Jumptap). Цей проект призвів до випуску специфікації OpenRTB Mobile 1.0, яка була опублікована у лютому 2011 року. Через широку популярність у галузі, OpenRTB був прийнятий як стандарт IAB у січні 2012 з версією 2.1.

Базовий протокол між учасниками аукціону є HTTP. Зокрема, для надсилання запитів на пропозиції використовується HTTP POST, щоб забезпечити можливість передачі більших обсягів даних, ніж при використанні HTTP GET. Сповіщення про перемогу в аукціоні можуть бути виконані або за допомогою HTTP POST, або за допомогою HTTP GET на розсуд обмінника. Виклики, які повертають контент (наприклад, будь-яка відповідь на пропозицію), повинні повертати HTTP-код 200. Виклики, які не повертають контент у відповідь на коректні запити (наприклад, пуста відповідь на пропозицію, що є одним із варіантів позначення відсутності пропозиції), повинні повертати HTTP 204. Некоректні виклики (наприклад, запит на пропозицію, який містить неправильні або пошкоджені дані) повинні повертати HTTP 400.

JSON (JavaScript Object Notation) є рекомендованим форматом для передачі даних пропозицій і відповідей на пропозиції. JSON був обраний через поєднання зручності для сприйняття і компактності. Додатково,

обмінник може пропонувати також бінарні представлення (наприклад, стиснений JSON, ProtoBuf, Avro та ін.), які можуть бути більш ефективними з точки зору часу передачі та пропускної здатності.

У запиті вказується формат представлення за допомогою заголовка HTTP Content-Type. Тип мультимедійного представлення для стандартного JSON-представлення – "application/json". Формат відповіді повинен бути таким же, як формат запиту. Якщо використовуються альтернативні бінарні представлення, обмінник або SSP повинні вказати відповідний Content-Type. Наприклад: "Content-Type: avro/binary" або "Content-Type: application/x-protobuf".

Наведена нижче модель об'єкта використовується для запиту на пропозицію. Об'єкт верхнього рівня (у JSON це безіменний зовнішній об'єкт) позначається як BidRequest. Із його безпосередніх підоб’єктів, технічно обов'язковий лише Imp, оскільки він є основним для опису продажу показу та вимог до нього. Він вимагає принаймні одного з таких об'єктів: Banner (який може містити кілька форматів), Video, Audio та Native, для визначення типу показу.

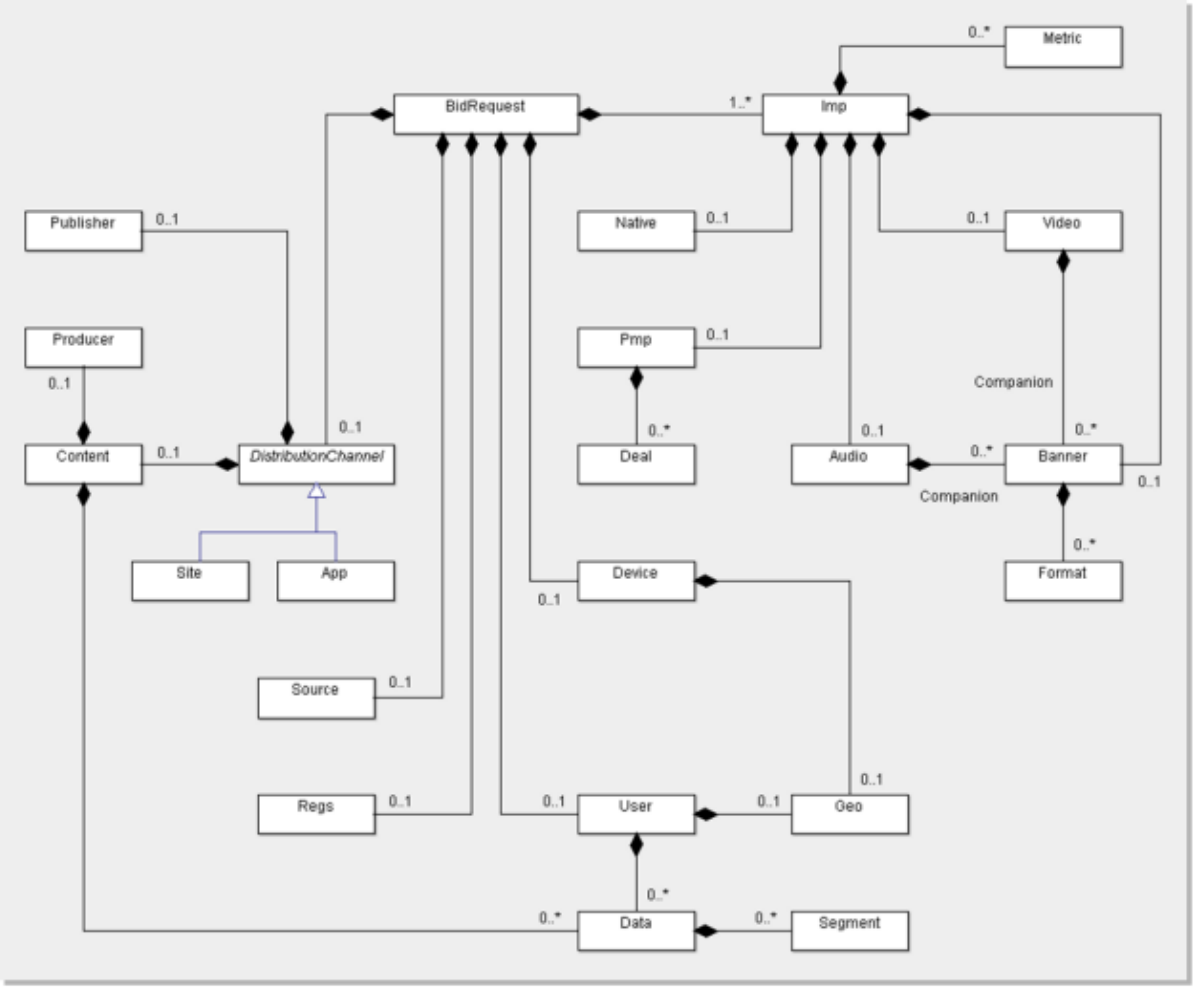

Рис. 1.4. Модель об’єкту BidRequest.

Інші елементи BidRequest надають різні форми інформації, яка допомагає учасникам аукціону в прийнятті рішень щодо цільового визначення та ціноутворення. Це включає дані про користувача, пристрій, яким він користується, місцезнаходження, регуляторні обмеження, контент та медіа, в яких відбудеться показ. Щодо останнього, існує різниця між сайтом та додатком (тобто програмою, зазвичай на мобільному пристрої). Абстрактний клас, що називається DistributionChannel, є лише концепцією моделювання, яка вказує, що BidRequest пов'язаний або з сайтом, або з додатком, але не з обома одночасно (тобто розподільний канал є абстракцією сайту та додатку). Як сайти, так і додатки можуть бути додатково описані даними про видавця, контент і його виробника.

На зображенні не показано об'єкт розширень (extensions). Це об'єкт невизначеної структури, який може бути доданий до будь-якого іншого об'єкта для передачі конкретних розширень обміну за стандартом. Обмінники, що використовують ці об'єкти, несуть відповідальність за публікацію своїх розширень для учасників аукціону.


```
{
    "id": "80ce30c53c16e6ede735f123ef6e32361bfc7b22",
    "at": 1, "cur": [ "USD" ],
    "imp": [
        {
            "id": "1", "bidfloor": 0.03,
            "banner": {
                "h": 250, "w": 300, "pos": 0
            }
        }
    ],
    "site": {
        "id": "102855",
        "cat": [ "IAB3-1" ],
        "domain": "www.foobar.com",
        "page": "http://www.foobar.com/1234.html ",
        "publisher": {
            "id": "8953", "name": "foobar.com",
            "cat": [ "IAB3-1" ],
            "domain": "foobar.com"
        }
    },
    "device": {
```


Рис. 1.5. Приклад BidRequest.

Наведена нижче модель об'єкта відповіді на ставку. Об'єкт верхнього рівня (тобто безіменний зовнішній об'єкт у форматі JSON) позначається як BidResponse у моделі. Відповідь на ставку може містити ставки від кількох "місць" (тобто сутності, що купує товар вище від самого учасника аукціону). Фактично, відповідь може містити кілька ставок від одного і того ж "місця"; зазвичай, але не обов'язково, від різних кампаній. Це може покращити шанси "місця" на перемогу, оскільки більшість обмінників забезпечують різні списки блокування від імені своїх видавців.

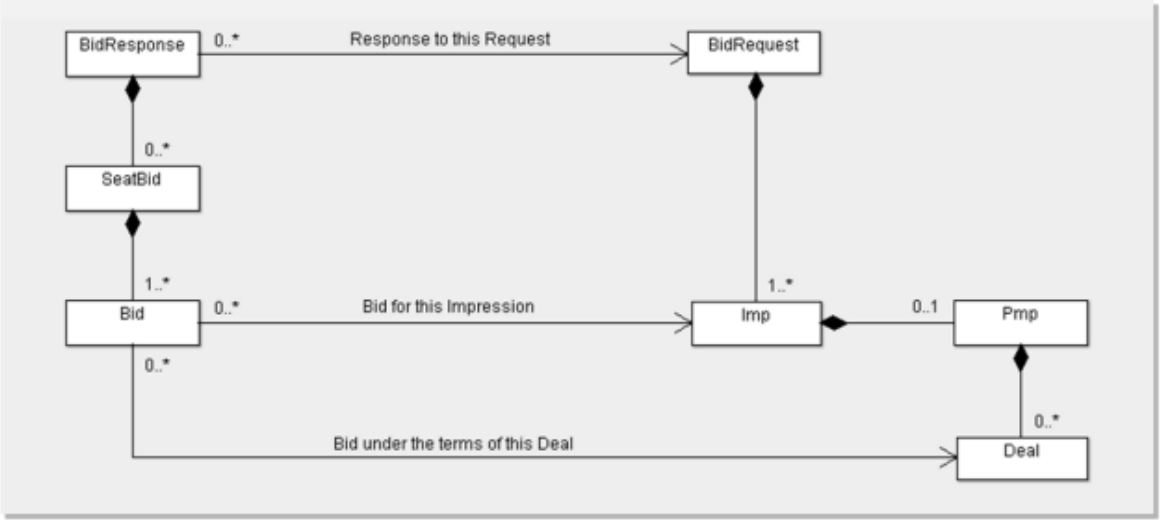

Рис. 1.6. Модель об’єкту BidResponse.

Фактичні об'єкти відповіді показані зліва, зокрема, об'єкт верхнього рівня BidResponse та колекції seat-специфічних об'єктів SeatBid з об'єктами Bid. Інші показані об'єкти є об'єктами з запиту на ставку, до яких відносяться об'єкти відповіді. Зокрема, BidResponse містить ID BidRequest для позитивного відстеження, і оскільки запит може містити кілька вражень, Bid містить ID Imp, для якого ставка є пропозицією до купівлі. Якщо ставка здійснюється в рамках угоди приватного ринку, Bid також включає ID конкретного об'єкта Deal. Не показано на зображенні об'єкт розширень. Це об'єкт не визначеної структури, який можна додати до будь-якого іншого об'єкта для передачі розширень, специфічних для учасника аукціону, відповідно до стандарту. Учасники аукціону, які використовують ці об'єкти, відповідають за публікацію своїх розширень до своїх обмінників.

```
{
   "id": "123",
   "seatbid": [
      {
         "bid": [
            {
               "id": "12345", "impid": "2", "price": 3.00,
               "nurl": "http://example.com/winnoticeurl",
               "adm": "<?xml version=\"1.0\" encoding=\"utf-8\"?>\n<VAST
version=\"2.0\">\n<Ad id=\"12345\">\n<InLine>\n<AdSystem
version=\"1.0\">SpotXchange</AdSystem>\n<AdTitle>\n <![CDATA[Sample
VAST]]>\n</AdTitle>\n<Impression>http://sample.com</Impression>\n<Description>\n<![C
DATA[A sample VAST feed]]>\n</Description>\n <Creatives>\n<Creative sequence=\"1\"
id=\"1\">\n<Linear>\n<Duration>00:00:30</Duration>\n  <TrackingEvents>
</TrackingEvents>\n<VideoClicks>\n<ClickThrough>\n<![CDATA[http://sample.com/openrtb
test]]>\n</ClickThrough>\n</VideoClicks>\n<MediaFiles>\n<MediaFile
delivery=\"progressive\" bitrate=\"256\" width=\"640\" height=\"480\"
type=\"video/mp4\">\n<![CDATA[http://sample.com/video.mp4]]>\n
</MediaFile>\n</MediaFiles>\n</Linear>\n
</Creative>\n</Creatives>\n</InLine>\n</Ad>\n</VAST>"
            }
         ]
      }
   ]
}
```


Рис. 1.7. Приклад BidResponse.

1.3. SSP платформи.

SSP (supply side platform) – це рекламна технологія, яка використовується видавцями для автоматизації продажу рекламного простору на веб-сайтах та додатках на основі переглядів. Значення SSP полягає в його роботі по з'єднанню видавців з рекламними мережами, платформами управління даними, рекламними біржами та DSP для продажу рекламного інвентарю.

В межах SSP рекламодавці можуть встановлювати фільтри, такі як рекламодавець, формат реклами, цільова аудиторія, тарифи на рекламний простір та інші, щоб продати свій рекламний інвентар.

SSP може бути довірена для автоматизації кількох процесів для видавця. Нижче ми розглянемо найпоширеніші процеси:

- **Оптимізація роботи з рекламними мережами**

  Оскільки у кожної рекламної мережі можуть бути власні вимоги, SSP використовує історичні дані для визначення, яку мережу використовувати, коли доступні покази на веб-сайті видавця.

Наприклад, одна мережа може платити $3.00 за 1000 вражень, але заповнювати рекламу лише на 20% часу, тоді як інша може платити $1.50 і заповнювати рекламу на 70% часу. Таким чином, видавці мають більшу контроль над своїм інвентарем, оскільки вони можуть бачити, які мережі платять за їх інвентар на рівні показів і відповідно оптимізувати цей процес.

- **Транзакції реального часу (real-time bidding)**

  З використанням рекламної біржі, платформа постачальника автоматизує продаж інвентару видавця платформам попиту (DSP).

- **Релевантність**

  За допомогою платформи постачальника, видавець може забезпечити відображення рекламних мереж, які є відповідними для доступного рекламного простору. Наприклад, видавець може використовувати платформу постачальника, щоб приймати враження лише з певних географічних регіонів.

- **Обмеження частоти показу**

  Щоб переконатися, що користувач додатку не бачить одну й ту ж рекламу занадто часто, платформи постачальників (SSP) і платформи попиту (DSP) співпрацюють, встановлюючи обмеження частоти показу. Це процес запису кількості разів, коли певна реклама показується конкретному відвідувачу, і подальше обмеження кількості показів цієї реклами цьому відвідувачу. Наприклад, якщо встановлено обмеження частоти показу на 3, то одна реклама може бути показана користувачу додатку не більше ніж 3 рази. Після досягнення цього обмеження, платформа постачальника і платформа попиту припиняють показ цієї реклами цьому конкретному користувачу, щоб уникнути перенасиченості однією рекламою та забезпечити більш різноманітне користувацьке досвід. Обмеження частоти показу дозволяє оптимізувати ефективність рекламних кампаній, забезпечуючи більш

рівномірний розподіл показів реклами серед різних користувачів і запобігаючи надмірному навантаженню на одного користувача.

Існує багато переваг використання SSP. Розглянемо п'ять найважливіших з них:

1. **Автоматизований продаж реклами** – очевидною перевагою є автоматизація продажу рекламного інвентару. SSP дозволяє уникнути ручної роботи, що підвищує ефективність процесу.
2. **Звітність** – іншою перевагою SSP є наявність звітності для видавців. SSP надають видавцям інформацію про цінність їх інвентарю для різних рекламодавців, оскільки вони можуть бачити, хто робить ставки, скільки заплачено за їх інвентар та який обсяг купують певні рекламодавці. Видавці можуть використовувати ці детальні дані для оптимізації з метою отримання максимального прибутку.
3. **Агрегування кількох мереж** – SSP також з'єднує видавців з кількома мережами, рекламними біржами та DSP, що дозволяє більшій кількості покупців брати участь у процесі миттєвих торгів. Це збільшує шанси на продаж рекламного простору видавців.
4. **Динамічне формування мінімальної цінової планки** – сьогодні стандартний SSP повинен дозволяти видавцям встановлювати м'які і жорсткі мінімальні цінові планки за CPM (вартість за 1000 показів). Ці цінові планки гарантують, що інвентар не буде продаватися нижче встановленої ціни. Це дозволяє проводити динамічні аукціони за моделями першої та другої ціни.
5. **Управління кампаніями та інвентарем** – SSP-реклама дозволяє видавцям керувати різними видами інвентарю, такими як банерна, відео, нативна реклама тощо, шляхом внесення рекламодавців до списку заборонених або дозволених, встановлення категорій IAB або конкретних каналів чи покупців, а також блокування певних типів реклами. Це дозволяє видавцям точно керувати тим, яку рекламу

вони дозволяють на своєму інвентарі, забезпечуючи вищу якість та відповідність реклами до їхньої аудиторії та контенту.

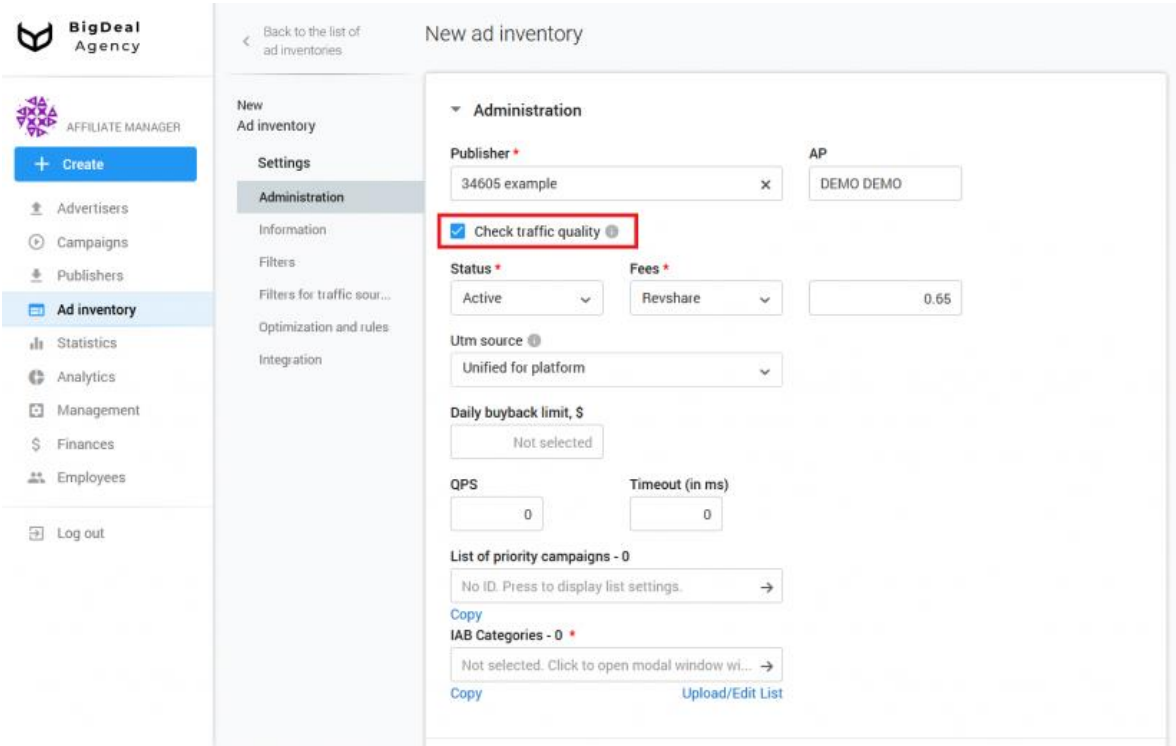

Рис. 1.8. Вигляд інтерфейсу SSP.

Для оцінки ефективності SSP існують наступні ключові показники продуктивності (KPI):

- **Дохід**: Загальний дохід, отриманий від вашого рекламного інвентару. Ефективна SSP повинна допомогти збільшити ваш дохід, підключаючи вас до високоякісних джерел попиту і оптимізуючи ваш інвентар.
- **Fill Rate (Відсоток заповнення):** Відсоток рекламних показів, які успішно продані та заповнені рекламою. Вищий показник заповнення вказує на те, що SSP успішно підключає ваш інвентар до відповідних джерел попиту та максимізує використання вашого рекламного простору.
- **Ефективна вартість за тисячу показів (eCPM):** Середній дохід, зароблений за тисячу рекламних показів. Вищий eCPM свідчить про те,

що SSP оптимізує вартість вашого рекламного інвентару та досягає кращих результатів у монетизації.

- **Вартість за тисячу показів (CPM):** Середня ціна, яку рекламодавці платять за тисячу рекламних показів. Вищий CPM свідчить про те, що SSP ефективно стимулює конкуренцію за ваш рекламний інвентар та максимізує дохід.
- **Якість рекламного інвентарю:** Якість доступного рекламного інвентарю на веб-сайті або додатку, включаючи видимість, безпеку бренду та розміщення реклами. Вищий якісний інвентар приваблює преміальних рекламодавців та генерує кращий дохід.
- **Різноманітність джерел попиту:** Кількість різних джерел попиту, таких як DSP, рекламні біржі та рекламодавці, підключених до SSP. Різноманіття джерел попиту збільшує конкуренцію за рекламний інвентар та допомагає максимізувати дохід.
- **Звітність та аналітика:** Якість та деталізованість звітності та аналітики, наданої SSP. Комплексні інструменти звітності дозволяють видавцям приймати рішення на основі даних та оптимізувати свої стратегії монетизації.
- **Функціональність та можливості платформи:** Функції та можливості SSP, такі як підтримка хедерного аукціону, розширені параметри таргетингу та технологія реального часу. Ефективна SSP повинна надавати ряд інструментів та функцій, які допомагають видавцям максимізувати вартість їх рекламного інвентару.

Оцінка цих ключових показників продуктивності допоможе оцінити ефективність вашої SSP та визначити можливості для оптимізації та покращення монетизації рекламного інвентарю.

2. РОЗРОБКА ПРОЕКТУ

2.1. Структура проекту.

Проект розроблено за допомогою фреймворку React мови JavaScript. Структура файлів зображена нижче:

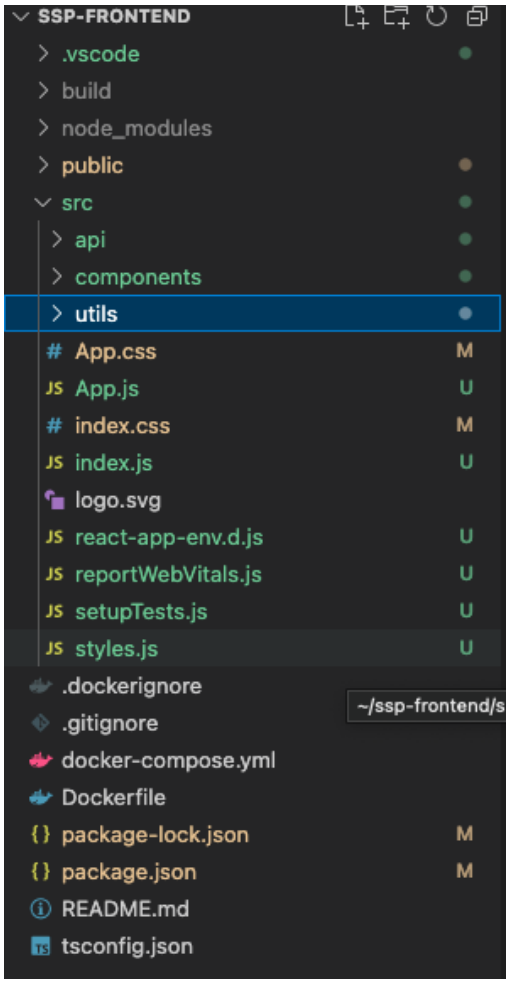

Рис. 2.1. Структура проекту.

Каталог build містить «збірку» додатка, яка об’єднує всі файли CSS в один файл і робить те саме з JavaScript файлами.

В каталозі node_modules зберігаються всі завантажених пакетів з NPM, необхідні для проекту.

Каталог public містить статичні файли, такі як index.html, файли бібліотек JavaScript, зображення, іконки і т.д. Ці статичні файли не обробляються webpack — пакувальником статичних модулів додатків JavaScript. Каталог public корисний як обхідний шлях у деяких випадках, наприклад, коли треба

включити невеликий скрипт за межі пакетного коду. Деякі бібліотеки взагалі можуть бути несумісними з webpack, і тоді можна лише включити їх як тег <script>.

Також проект має кілька важливих файлів:

1. .gitignore — файл, який використовується системою контролю версій git. В цьому файлі вказуються назви файлів або папок проекту, які не треба додавати в комміт при завантаженні змін в репозиторій.

2. package.json — в цьому файлі зберігаються всі скрипти і залежності проекту.

```
package.json > ...
1  {
2    "name": "ca-project-react",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "@material-ui/core": "^4.12.4",
7      "@material-ui/data-grid": "^4.0.0-alpha.37",
8      "@material-ui/icons": "^4.11.3",
9      "@mui/x-data-grid": "^5.10.0",
10     "axios": "^0.27.2",
11     "prop-types": "^15.8.1",
12     "react": "^18.1.0",
13     "react-async": "^10.0.1",
14     "react-dom": "^18.1.0",
15     "react-paginate": "^8.1.3",
16     "react-router-dom": "^6.3.0",
17     "react-scripts": "^5.0.1"
18   },
       Debug
19   "scripts": {
20     "start": "react-scripts start",
21     "build": "react-scripts build",
22     "test": "react-scripts test --env=jsdom",
23     "eject": "react-scripts eject"
24   },
```

Рис. 2.2. Структура файлу package.json.

Файл package.json складається з:

- name — назву програми.

- version – поточної версії програми.

- “private”: true — налаштування, яке дозволяє уникнути випадкової публікації додатка як загальнодоступного пакета в npm.

- dependencies — зберігає всі необхідні для програми модулі і їхні версії.

- scripts — визначаються назви команд, які визначають певні дії з React додатком, наприклад build.

3. package-lock.json — містить точне дерево залежностей, яке встановлено в /node_modules.

В каталозі src міститься код програми. Стандартно в цій папці знаходиться основний компонент App.js, пов’язані з ним стилі (App.css), index.js, який забезпечує точку входу в додаток та його стилі index.css. Каталог src також містить три підкаталоги, в яких зберігається основний код проекту.

Каталог **api** містить файли, які дозволяють працювати із сервером, доступатись до даних та оновлювати їх. В рамках цієї роботи не передбачена розбробка серверної частини, тому дані зберігаються або в localStorage браузера, або створені вручну (наприклад, для візуалізації графіків). Каталог містить 5 Javascript файлів, які відповідають за дані різних частин додатку

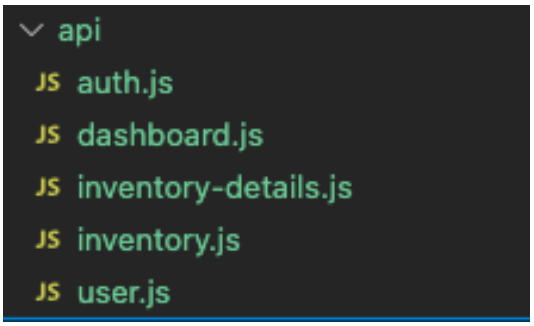

Рис 2.3. Структура каталогу api

Наступний каталог – **components.** В ньому зберігаються компоненти - компонентів — автономних, логічних фрагментів коду, які описують частину інтерфейсу користувача. Ці компоненти можна скласти разом, щоб створитиповноцінний інтерфейс користувача, а React абстрагує більшу частину роботи з візуалізації, залишаючи можливість сконцентруватися на дизайні інтерфейсу користувача. Компоненти можуть мати в собі певний функціонал (наприклад, CreateInventory, яка створює рекламні інвентарі), відповідати суто за вивід даних (InventoryList), або використовуватись для побудови структури інтерфейсу (Sidebar, Header).

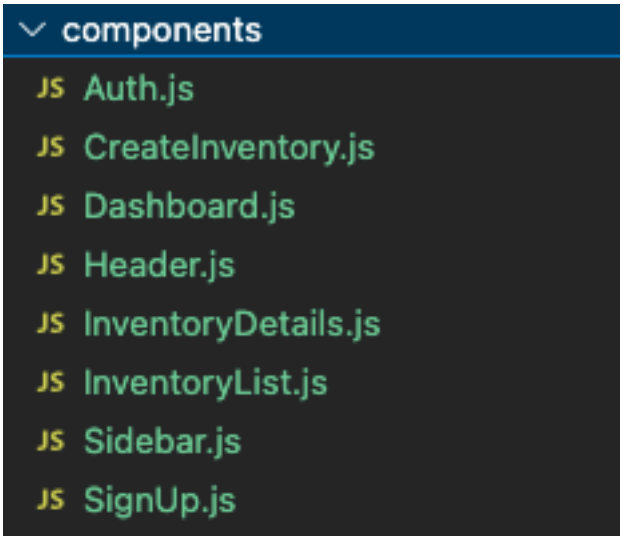

Рис. 2.4. Структура каталогу components

В каталозі **utils** зберігаються допоміжні функції для роботи з мережею або авторизацією.

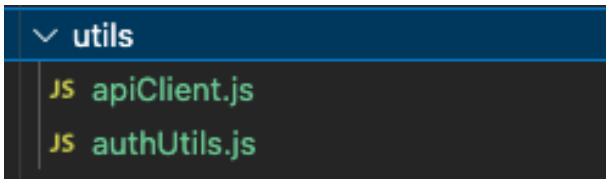

Рис. 2.5. Структура каталогу utils

2.2. Інтерфейс додатку

Для реєстрації та логіну користувача створено відповідні компоненти.

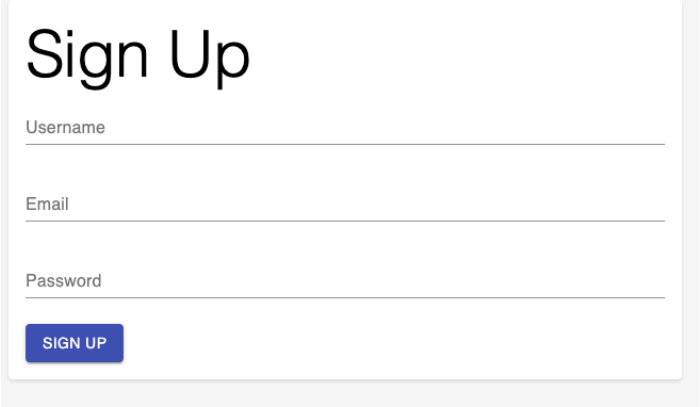

Рис. 2.6. Сторінка реєстрації користувача.

Для виводу загальної інформації по кількості запитів, показів реклами і натискань на неї буде використовуватись компонента Dashboard. Тут є графік цих показників.

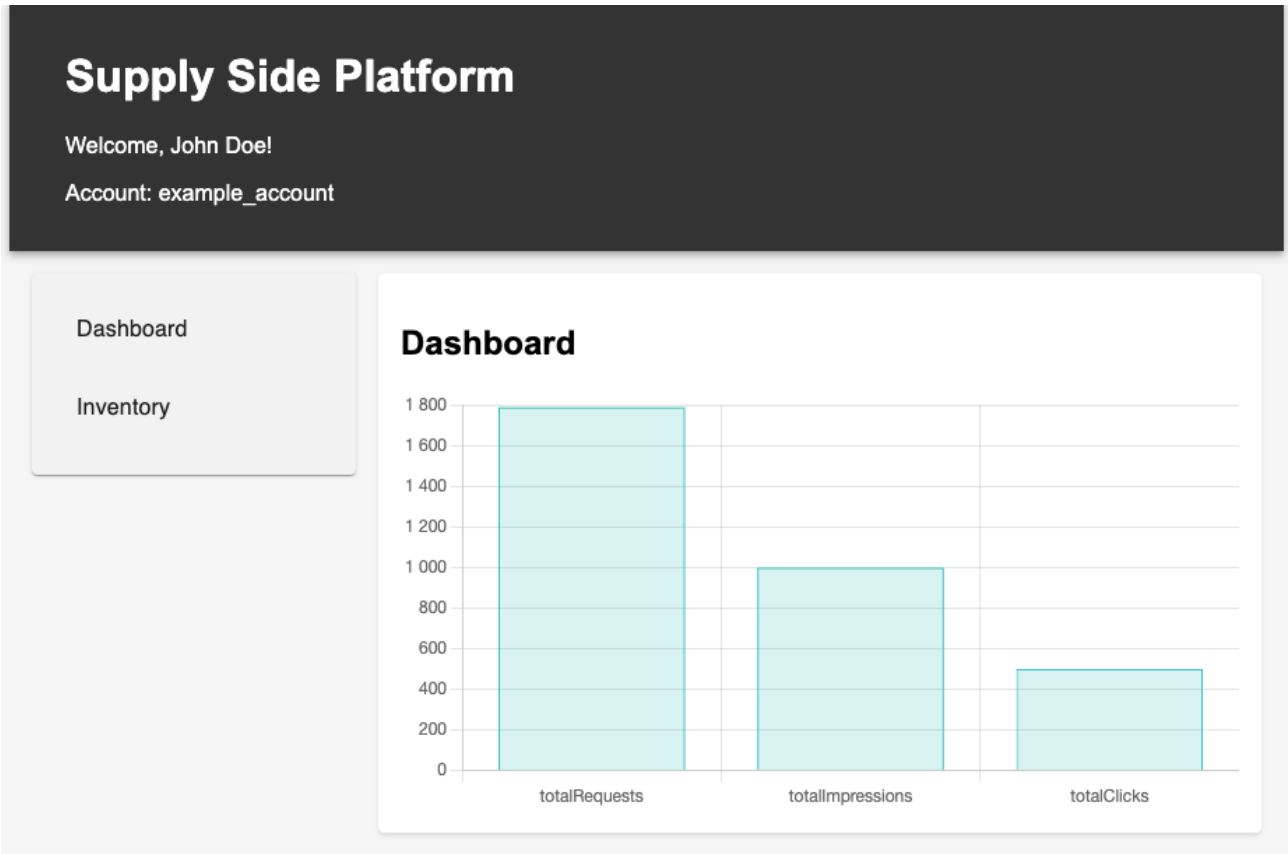

Рис. 2.7. Сторінка Dashboard і графік по загальній статистиці.

Натиснувши на вкладку Inventory користувач побачить список вже створених інвентарів та їх короткого опису (назву, домен та категорію).

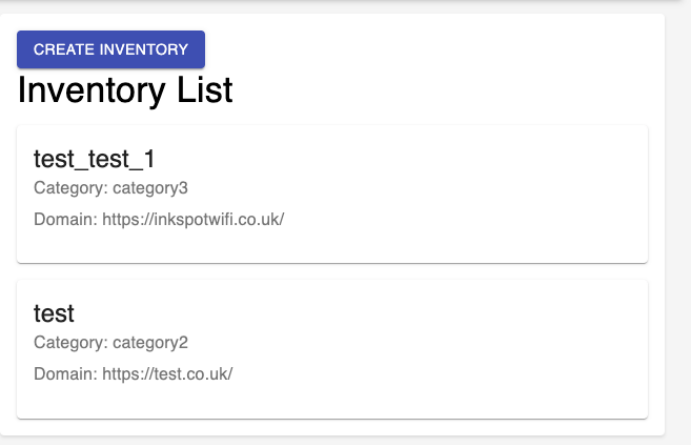

Рис. 2.8. Список інвентарів.

Для створення нового інвентаря користувач може натиснути на кнопку Create Inventory. Після цього на сторінці з’явиться форма, де він зможе задати основні поля для опису рекламного місця та додати розміри.

Рис. 2.9. Форма для створення інвентаря.

Натиснувши на назву інвентаря, користувач перейде на сторінку з деталями, де є інфомація про плейсменти та графік із погодинною статистикою по запитам на рекламу, показам та натисканням на неї.

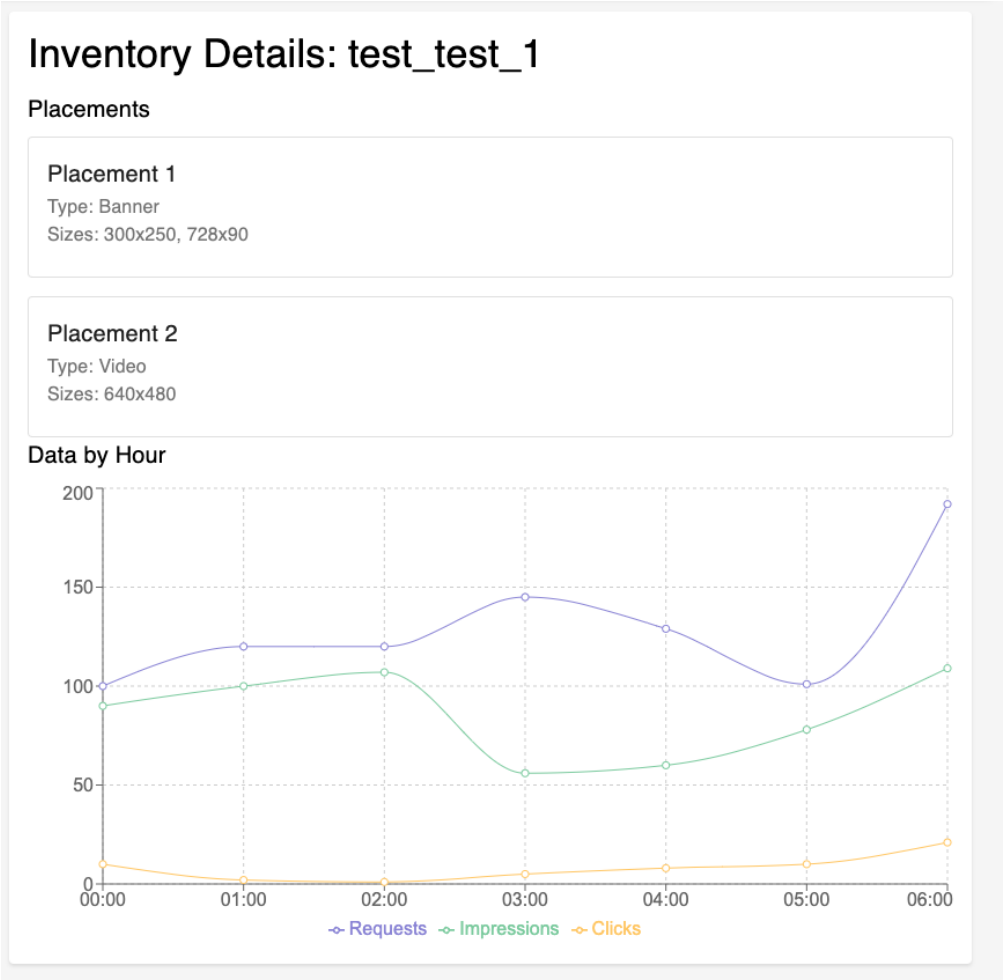

Рис. 2.10. Деталі інвентаря та графік зі статистикою.

ВИСНОВКИ

В роботі було розглянуто та проаналізовано теоретичні відомості про створення цифрової реклами за домопогою programmatic платформ, наведено відомості про те, за якою схемою і протоколом подібні платформи працюють. Також розглянуто поняття SSP та їх роль в програматичній рекламі. Було описано структурута функціонал програми, її інтерфейс.

При розробці були використані мова програмування JavaScript, бібліотека React та інтерфейсний фреймворк MaterialUI.

Результатом виконання курсової роботи «Розробка клієнтської частини SSP платформи» є програма, яка дозволяє створювати рекламні місця (інвентарі) та переглядати статистику. Програма має гарний інтерфейс, є зручною у використанні, не займає багато ресурсів та має хорошу швидкодію. Робота може бути використаною як частина магістерської роботи, в якій можна проаналізувати оптимальність цін на рекламні місця за допомогою штучного інтелекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. OpenRTB API Specification Version 2.5 — https://www.iab.com/wp-content/uploads/2016/03/OpenRTB-API-Specification-Version-2-5-FINAL.pdf
2. An overview of the Programmatic Advertising Ecosystem: Opportunities and Challenges — https://platformobservatory.eu/app/uploads/2019/10/An-overview-of-the-Programmatic-Advertising-Ecosystem-Opportunities-and-Challenges.pdf
3. What Is A Supply-Side Platform (SSP)? | Martech.zone — https://martech.zone/what-is-a-supply-side-platform-ssp/
4. Programmatic Advertising - The ultimate guide — https://www.bannerflow.com/resources/ultimate-guides/programmatic-advertising/
5. What Is Programmatic Advertising? How Does It Work? — https://www.searchenginejournal.com/what-is-programmatic-advertising/462655/
6. React docs — https://react.dev/learn
7. MaterialUI — Overview — https://mui.com/material-ui/getting-started/overview/
8. JavaScript | MDN — https://developer.mozilla.org/ru/docs/Web/JavaScript

ДОДАТКИ

Додаток 1

Код файлу Auth.js

```
import React, { useState } from 'react';

import { TextField, Button, Typography, Container, Paper } from '@material-ui/core';

const AuthPage = () => {

  const [email, setEmail] = useState('');

  const [password, setPassword] = useState('');

  const handleEmailChange = (e) => {

    setEmail(e.target.value);

  };

  const handlePasswordChange = (e) => {

    setPassword(e.target.value);

  };

  const handleSubmit = (e) => {

    e.preventDefault();

    // Perform authentication logic here

    console.log('Email:', email);

    console.log('Password:', password);

  };

  return (

    <Container maxWidth="sm">

      <Paper elevation={3} style={{ padding: '16px', marginTop: '100px' }}>

        <Typography variant="h5" align="center" gutterBottom>

          Authentication
```

Продовження дод. 1

```
</Typography>
      <form onSubmit={handleSubmit}>
        <TextField
          type="email"
          label="Email"
          fullWidth
          margin="normal"
          value={email}
          onChange={handleEmailChange}
        />
        <TextField
          type="password"
          label="Password"
          fullWidth
          margin="normal"
          value={password}
          onChange={handlePasswordChange}
        />
        <Button type="submit" variant="contained" color="primary" fullWidth>
          Login
        </Button>
      </form>
    </Paper>
  </Container>
);}; export default AuthPage;
```

Додаток 2

Код файлу CreateInventory.js

```
import React, { useState } from 'react';

import { Typography, TextField, Button, FormControl, InputLabel, Select, MenuItem, Box, Grid } from '@material-ui/core';

import { createInventory } from '../api/inventory';

import '../App.css';



const CreateInventory = () => {

  const [inventoryData, setInventoryData] = useState({

    name: '',

    iabCategory: '',

    language: '',

    price: '',

    location: '',

    domain: '',

    placements: [],

  });



  const [placementData, setPlacementData] = useState({

    sizes: [],

    type: ''

  });



  const handleChange = (e) => {

    const { name, value } = e.target;

    setInventoryData((prevData) => ({ ...prevData, [name]: value }));  };
```

Продовження дод. 2

```
const handlePlacementChange = (e) => {
  const { name, value } = e.target;
  setPlacementData((prevData) => ({ ...prevData, [name]: value }));
};

const handleSizeChange = (e, index) => {
  const { value } = e.target;
  setPlacementData((prevData) => {
    const sizes = [...prevData.sizes];
    sizes[index] = value;
    return { ...prevData, sizes };
  });
};

const handleAddPlacement = () => {
  const newPlacement = { ...placementData };
  setInventoryData((prevData) => ({
    ...prevData,
    placements: [...prevData.placements, newPlacement],
  }));
  setPlacementData({
    name: '',
    sizes: [],
  });
};
```

Продовження дод. 2

```
const handleRemovePlacement = (index) => {
  setInventoryData((prevData) => {
    const placements = [...prevData.placements];
    placements.splice(index, 1);
    return { ...prevData, placements };
  });
};


const handleSubmit = async (e) => {
  e.preventDefault();


  try {
    // Create inventory
    await createInventory(inventoryData);


    // Clear form fields
    setInventoryData({
      name: '',
      iabCategory: '',
      language: '',
      format: '',
      price: '',
      location: '',
      placements: [],
    });
```

Продовження дод. 2

```
setPlacementData({
      name: '',
      sizes: [],
    });

    // Show success message or perform any other desired action
    console.log('Inventory created successfully!');
  } catch (error) {
    // Handle error appropriately
    console.error('Failed to create inventory:', error.message);
  }
};

return (
  <div>
    <Typography variant="h4" gutterBottom>
      Create Inventory
    </Typography>

    <form onSubmit={handleSubmit}>
      <Box display="flex" flexDirection="column" marginLeft="10px">
        <Grid container spacing={2}>
          <Grid item xs={12}>
            <TextField
              name="name"
```

Продовження дод. 2

```
label="Inventory Name"
        value={inventoryData.name}
        onChange={handleChange}
        margin="normal"
        required
      />
    </Grid>
    <Grid item xs={12}>
      <FormControl margin="normal">
        <InputLabel htmlFor="iab-category">IAB Category</InputLabel>
        <Select
          id="iab-category"
          name="iabCategory"
          value={inventoryData.iabCategory}
          onChange={handleChange}
          required
        >
          <MenuItem value="IAB-1">IAB-1</MenuItem>
          <MenuItem value="IAB-2">IAB-2</MenuItem>
          <MenuItem value="IAB-3">IAB-3</MenuItem>
          <MenuItem value="IAB-4">IAB-4</MenuItem>
          <MenuItem value="IAB-5">IAB-5</MenuItem>
        </Select>
      </FormControl>
    </Grid>
```

Продовження дод. 2

```
<Grid item xs={12}>
        <FormControl margin="normal">
          <InputLabel htmlFor="language">Language</InputLabel>
          <Select
            id="language"
            name="language"
            value={inventoryData.language}
            onChange={handleChange}
            required
          >
            <MenuItem value="english">English</MenuItem>
            <MenuItem value="spanish">Spanish</MenuItem>
            <MenuItem value="french">French</MenuItem>
          </Select>
        </FormControl>
      </Grid>
      <Grid item xs={12}>
        <TextField
          name="domain"
          label="Site domain"
          value={inventoryData.domain}
          onChange={handleChange}
          margin="normal"
          required
        />
```

Продовження дод. 2

```
</Grid>
      <Grid item xs={12}>
        <TextField
          name="price"
          label="Price"
          type="number"
          value={inventoryData.price}
          onChange={handleChange}
          margin="normal"
          required
        />
      </Grid>
      <Grid item xs={12}>
        <TextField
          name="location"
          label="Location"
          value={inventoryData.location}
          onChange={handleChange}
          margin="normal"
          required
        />
      </Grid>
    </Grid>


    <Typography variant="h6" gutterBottom>
```

Продовження дод. 2

```
Placements
      </Typography>
      {inventoryData.placements.map((placement, index) => (
        <div key={index}>
          <Typography variant="subtitle1">Placement {index + 1}</Typography>
          <Grid container spacing={2}>
            <Grid item xs={12} sm={6}>
              <FormControl margin="normal">
                <InputLabel htmlFor={`type_${index}`}>Type</InputLabel>
                <Select
                  id={`type_${index}`}
                  name={`type_${index}`}
                  value={placement.type}
                  onChange={handlePlacementChange}
                  required
                >
                  <MenuItem value="banner">Banner</MenuItem>
                  <MenuItem value="video">Video</MenuItem>
                  <MenuItem value="native">Native</MenuItem>
                </Select>
              </FormControl>
            </Grid>
            <Grid item xs={12} sm={6}>
              {placement.sizes.map((size, sizeIndex) => (
                <TextField
```

Продовження дод. 2

```
key={sizeIndex}
                label={`Size ${sizeIndex + 1}`}
                name={`size_${index}_${sizeIndex}`}
                value={size}
                onChange={handleSizeChange}
                margin="normal"
                required
               />
             ))}
           </Grid>
          </Grid>
          <Button variant="outlined" color="secondary" onClick={() => handleRemovePlacement(index)}>
            Remove Placement
          </Button>
        </div>
      ))}

      <Typography variant="h6" gutterBottom>
        Add Placement
      </Typography>
      <Grid container spacing={2}>
        <Grid item xs={12} sm={6}>
          <FormControl margin="normal">
            <InputLabel htmlFor="type">Type</InputLabel>
            <Select
```

Продовження дод. 2

```
id="type"
            name="type"
            value={placementData.type}
            onChange={handlePlacementChange}
            required
          >
            <MenuItem value="banner">Banner</MenuItem>
            <MenuItem value="video">Video</MenuItem>
            <MenuItem value="native">Native</MenuItem>
          </Select>
        </FormControl>
      </Grid>
      <Grid item xs={12} sm={6}>
        {placementData.sizes.map((size, index) => (
          <TextField
            key={index}
            label={`Size ${index + 1}`}
            name={`size_${index}`}
            value={size}
            onChange={(e) => handleSizeChange(e, index)}
            margin="normal"
            required
          />
        ))}
      </Grid>
```

Продовження дод. 2

```
</Grid>
        <Grid item xs={12} sm={6}>
          <Button variant="outlined" color="primary" onClick={handleAddPlacement}>
            Add Placement
          </Button>
        </Grid>
        <Grid item xs={12} sm={6}>
          <Button type="submit" margin='20px' variant="contained" color="primary">
            Create Inventory
          </Button>
          </Grid>
      </Box>
    </form>
  </div>

 );
};

export default CreateInventory;
```

Додаток 3

Код файлу Dashboard.js

```
import React, { useEffect, useState } from 'react';

import { fetchDashboardData } from '../api/dashboard';

import { Chart, BarController, BarElement, CategoryScale, LinearScale } from 'chart.js';



Chart.register(BarController, CategoryScale, LinearScale, BarElement);



const Dashboard = () => {

  const [statistics, setStatistics] = useState([]);



  useEffect(() => {

    const fetchStatistics = async () => {

      try {

        const statisticsData = await fetchDashboardData()

        setStatistics(statisticsData);

        createChart(statisticsData);

      } catch (error) {

        console.error('Failed to fetch statistics:', error);

      }

    };



    fetchStatistics();

  }, []);



  const createChart = (statisticsData) => {
```

Продовження дод. 3

```
const ctx = document.getElementById('chart').getContext('2d');
  const labels = Object.keys(statisticsData);
  const values = Object.values(statisticsData);
  new Chart(ctx, {
    type: 'bar',
    data: {
      labels: labels,
      datasets: [
        {
          label: 'Statistics',
          data: values,
          backgroundColor: 'rgba(75, 192, 192, 0.2)',
          borderColor: 'rgba(75, 192, 192, 1)',
          borderWidth: 1,
        },
      ],
    },
    options: {
      scales: {
        y: {
          beginAtZero: true,
        },
      },
      plugins: {
        legend: {
```

Продовження дод. 3

```
display: true,
            position: 'right',
            labels: {
              usePointStyle: true,
            },
          },
        },
      }
    });
  };

  return (
    <div className='content-view'>
      <h2>Dashboard</h2>
      <canvas id="chart" />
    </div>
  );
};

export default Dashboard;
```

Додаток 4

Код файлу InventoryDetails.js

```
import { useState, useEffect } from 'react';

import { useParams } from 'react-router-dom';

import { Typography, Grid, Card, CardContent } from '@material-ui/core';

import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, Legend } from 'recharts';

import getInventoryDetails from '../api/inventory-details';



const InventoryDetails = () => {

  const { inventoryName } = useParams();

  const [inventory, setInventory] = useState(null);

  const [data, setData] = useState([]);



  useEffect(() => {

    const fetchData = async () => {

      const inventoryDetails = await getInventoryDetails(inventoryName);

      setInventory(inventoryDetails.inventory);

      setData(inventoryDetails.data);

    };



    fetchData();

  }, [inventoryName]);



  return (

    <div>

      {inventory && (
```

Продовження дод. 4

```
<div>
    <Typography variant="h4" gutterBottom>
      Inventory Details: {inventory.name}
    </Typography>

    <Typography variant="h6" gutterBottom>
      Placements
    </Typography>

    <Grid container spacing={2}>
      {inventory.placements.map((placement) => (
        <Grid item xs={12} sm={6} md={12}>
          <Card key={placement.id} variant="outlined">
          <CardContent margin="20px">
            <Typography variant="h6">{placement.name}</Typography>
            <Typography color="textSecondary">Type: {placement.type}</Typography>
            <Typography color="textSecondary">Sizes: {placement.sizes.join(', ')}</Typography>
          </CardContent>
        </Card>
        </Grid>
      ))}
    </Grid>

    <Typography variant="h6" gutterBottom>
      Data by Hour
```

Продовження дод. 4

```
</Typography>
        <Grid container spacing={2}>
          <Grid item xs={12} sm={6} md={8}>
            <LineChart width={800} height={400} data={data}>
              <CartesianGrid strokeDasharray="3 3" />
              <XAxis dataKey="hour" />
              <YAxis />
              <Tooltip />
              <Legend />
              <Line type="monotone" dataKey="requests" name="Requests" stroke="#8884d8" />
              <Line type="monotone" dataKey="impressions" name="Impressions" stroke="#82ca9d" />
              <Line type="monotone" dataKey="clicks" name="Clicks" stroke="#ffc658" />
            </LineChart>
          </Grid>
        </Grid>
      </div>
    )}
  </div>
 );
};


export default InventoryDetails;
```

Додаток 5

Код файлу InventoryList.js

```
import { getInventory } from '../api/inventory';

import { useState, useEffect } from 'react';

import { Button, Grid, Card, CardContent, Typography } from '@material-ui/core';

import CreateInventory from './CreateInventory';

import '../App.css';

import { useNavigate } from 'react-router-dom';



const InventoryList = () => {

  const navigate = useNavigate();

  const [inventories, setInventories] = useState([]);

  const [isShowCreateInventory, setShowCreateInventory] = useState(false);



  useEffect(() => {

    const fetchData = async () => {

      const fetchedInventories = await getInventory();

      setInventories(fetchedInventories);

    };



    fetchData();

  }, []);



  const handleCreateInventory = () => {

    setShowCreateInventory(true);

  };
```

Продовження дод. 5

```
const handleCloseInventory = () => {
  setShowCreateInventory(false);
}


const handleInventoryClick = (inventoryName) => {
  navigate(`/inventory/${inventoryName}`);
};


return (
  <div className="content-view">
    <Button variant="contained" color="primary" onClick={handleCreateInventory}>
      Create Inventory
    </Button>
    {isShowCreateInventory ? <CreateInventory /> : null}
    {isShowCreateInventory ? <Button onClick={handleCloseInventory}>Close</Button> : null}
    <Typography variant="h4" gutterBottom>
      Inventory List
    </Typography>
    {inventories.length === 0 ? (
      <Typography variant="body1">No inventories available.</Typography>
    ) : (
      <Grid container spacing={2}>
        {inventories.map((inventory) => (
          <Grid item xs={12} sm={6} md={12} key={inventory.name} onClick={() => handleInventoryClick(inventory.name)}>
            <Card>
```

Продовження дод. 5

```
<CardContent>
                <Typography variant="h5" component="h2">
                  {inventory.name}
                </Typography>
                <Typography color="textSecondary" gutterBottom>
                  Category: {inventory.iabCategory}
                </Typography>
                <Typography color="textSecondary" gutterBottom>
                  Domain: {inventory.domain}
                </Typography>
                {/* Add more inventory details here */}
              </CardContent>
            </Card>
          </Grid>
        ))}
      </Grid>
    )}
  </div>
 );
};


export default InventoryList;
```