МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ТАРАН НАЗАР МИКОЛАЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри інформаційних технологій,
Зелінська О.В.
«___» _____ 2024р.

РЕКОМЕНДАЦІЙНА СИСТЕМА ПЛАНУВАННЯ РЕКЛАМНОЇ КАМПАНІЇ НА ОСНОВІ АНАЛІЗУ ДАНИХ ВЕБТРАФІКУ ТА ПОВЕДІНКИ КОРИСТУВАЧІВ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (магістерська) робота

Науковий керівник:
Зелінська О.В., к.т.н.
доцент кафедри
інформаційних технологій
_____
(підпис)

Оцінка: _____ / _____ / _____

Голова ЕК: _____
(підпис)

Вінниця 2024

АНОТАЦІЯ

Таран Н.М. Рекомендаційна система планування рекламної кампанії на основі аналізу даних вебтрафіку та поведінки користувачів. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні технології обробки даних». Донецький національний університет імені Василя Стуса, Вінниця, 2024.

Дана кваліфікаційна робота розглядає рекомендаційну систему планування рекламної кампанії на основі аналізу даних вебтрафіку та поведінки користувачів. Проаналізовані існуючі системи планування рекламної кампанії. Було обґрунтовано вибір архітектури рекомендаційної системи та розглянуто технології для реалізації системи. Розроблено модель рекомендаційної системи, що складається з серверної частини, клієнтської частини та бази даних, реалізовано рекомендаційну систему планування рекламної кампанії та поведінки користувачів за допомогою мови програмування C#, з використанням технологій ASP.NET, Angular та MySQL.

Taran N.M. Recommendation system for planning an advertising campaign based on the analysis of web traffic data and user behavior. Specialty 122 "Computer science", educational program "Computer data processing technologies". Vasyl Stus Donetsk National University, Vinnytsia, 2024.

This qualifying work considers a recommender system for planning an advertising campaign based on the analysis of web traffic data and user behavior. Existing advertising campaign planning systems were analyzed. The choice of the architecture of the recommender system was substantiated, then the technologies for implementing the system were considered. A model of a recommender system consisting of a server part, a client part and a database was developed. A recommender system for planning an advertising campaign and user behavior was implemented using the C# programming language, using ASP.NET, Angular and MySQL technologies.

ЗМІСТ

ВСТУП 4

РОЗДІЛ 1 АНАЛІЗ СУЧАСНИХ СИСТЕМ ПЛАНУВАННЯ РЕКЛАМНОЇ КАМПАНІЇ 5

1.1 Задача планування рекламної кампанії в сучасних умовах 5

1.2 Особливості функціонування сучасних систем планування рекламної кампанії 7

1.3 Постановка задачі по створенню системи планування рекламної кампанії 10

РОЗДІЛ 2 ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ СИСТЕМИ ПЛАНУВАННЯ РЕКЛАМНОЇ КАМПАНІЇ 12

2.1 Архітектура рекомендаційної системи 12

2.2 Обгрунтування вибору технологій та інструментів для реалізації системи 14

2.3 Моделювання системи планування рекламної кампанії на основі аналізу даних вебтрафіку та поведінки користувачів 23

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ПЛАНУВАННЯ РЕКЛАМНОЇ КАМПАНІЇ НА ОСНОВІ АНАЛІЗУ ДАНИХ ВЕБТРАФІКУ ТА ПОВЕДІНКИ КОРИСТУВАЧІВ 43

3.1 Вимоги до системи планування рекламної кампанії 43

3.2 Програмна реалізація 44

3.3 Тестування та верифікація системи 66

ВИСНОВКИ 74

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 75

ВСТУП

У сучасному цифровому світі, де реклама відіграє ключову роль у просуванні товарів і послуг, ефективне планування рекламної кампанії стає все важливішим для підприємств та маркетологів. Залучення правильної аудиторії та максимізація результативності реклами вимагають глибокого розуміння поведінки користувачів та аналізу даних вебтрафіку.

Створення системи планування рекламної кампанії на основі аналізу даних вебтрафіку та поведінки користувачів є сучасним та перспективним напрямом розвитку маркетингових стратегій. Ця система поєднує в собі потужність аналітики даних, розуміння інтересів та потреб цільової аудиторії, а також можливість точного спрямування рекламних зусиль.

В основі такої системи лежить аналіз вебтрафіку, який надає важливі відомості про поведінку користувачів на вебсайті або інших цифрових платформах. Він включає в себе інформацію про переглянуті сторінки, взаємодію з контентом, конверсії та інші параметри, що дозволяють зрозуміти, як користувачі сприймають рекламний контент та взаємодіють з ним.

На основі зібраних даних вебтрафіку можна створити профіль цільової аудиторії, з'ясувати її інтереси, поведінку та вподобання. Це дозволить налаштувати рекламну кампанію таким чином, щоб досягти найкращих результатів і ефективно залучити цільову аудиторію.

Створення системи планування рекламної кампанії на основі аналізу даних веб-трафіку та поведінки користувачів є важливим етапом у вдосконаленні маркетингових стратегій, та досягненні успіху у конкурентному цифровому середовищі. Вона дозволяє маркетологам розуміти свою аудиторію краще, покращувати ефективність рекламних кампаній та досягати бажаних результатів.

Результати дослідження представлені на IV Всеукраїнській науково-практичній конференції «Комп’ютерні технології обробки даних».

РОЗДІЛ 1
АНАЛІЗ СУЧАСНИХ СИСТЕМ ПЛАНУВАННЯ РЕКЛАМНОЇ КАМПАНІЇ

1.1 Задача планування рекламної кампанії в сучасних умовах

В умовах розвитку сучасних інформаційних технологій та збільшення попиту різних продуктів на ринках виникає потреба оптимізації витрат на залучення нових споживачів та підтримку інтересу до продукту споживачів, які вже раніше купували продукт.

Для підтримки інтересу до продукту старих споживачів достатньо постійно, з певним інтервалом, коригувати залежність ціна-якість. Звісно, щоб не допустити перехід споживачів до конкурентів, необхідно також слідкувати за ринком та зважати на інші критерії вибору споживачем певного продукту, таких як обслуговування, підтримка споживачів, наявність гарантії (переважно для техніки) і т.д. Залучення нових споживачів може відбуватись за допомогою кількох методів:

- рекомендації продукту від друзів чи знайомих;
- надання безкоштовних пробників продукту;
- SEO оптимізація для пошукових систем;
- організація реферальних програм;
- реклама.

Реклама є основним методом для багатьох великих компаній як Apple, CocaCola, Lays, Wolkswagen і т.д. Реклама є важливим інструментом для просування продуктів, послуг та брендів, оскільки вона допомагає залучити увагу споживачів, створити свідомість про продукт і вирішити проблему попиту та пропозиції. Вона впливає на споживачів, впродовж часу підтримує бренд,

підсилює конкурентну боротьбу та сприяє економічному зростанню, допомагаючи підприємствам залучати нових клієнтів і розширювати свій бізнес.

Планування рекламної кампанії - це важливий етап у розробці ефективної рекламної стратегії, який передбачає ретельне визначення цілей, аудиторії, бюджету та засобів для досягнення успіху. Ось деякі основні особливості планування рекламної кампанії:

- Визначення цілей та мети: Першим кроком є чітке формулювання цілей рекламної кампанії. Це може бути збільшення усвідомленості бренду, збільшення продажів, залучення нових клієнтів, підвищення лояльності тощо. Мета повинна бути конкретною, вимірюваною, досяжною, реалістичною та часовою (модель SMART).
- Аналіз цільової аудиторії: Важливо ретельно дослідити та зрозуміти свою цільову аудиторію - хто вони, які їхні потреби, побажання та психографічні характеристики. Це допоможе вибрати найбільш ефективні канали та повідомлення для залучення їх уваги.
- Вибір рекламних каналів: Враховуючи цільову аудиторію та бюджет, потрібно визначити найкращі канали реклами. Це можуть бути соціальні медіа, телебачення, радіо, інтернет-реклама, публікації у пресі або інші засоби залучення уваги.
- Розробка повідомлення і творчого підходу: Важливо створити рекламні матеріали, які відповідають цілям і цільовій аудиторії. Креативний підхід, який зацікавить та емоційно залучить споживачів, грає ключову роль у вдалій рекламній кампанії.
- Бюджетування і контроль витрат: Планування рекламної кампанії передбачає розробку бюджету і контроль витрат. Бюджет повинен бути достатнім для досягнення поставлених цілей, а також розподілений між різними каналами та етапами кампанії.

- Визначення ключових показників ефективності (KPI): Для оцінки результатів кампанії потрібно встановити KPI, які відображаються на досягненні цілей. Це можуть бути показники, такі як кількість переходів на сайт, конверсія, ROI (прибуток від інвестицій) тощо.
- Моніторинг та оптимізація: Рекламну кампанію слід постійно відстежувати та аналізувати результати, вносячи корективи, якщо необхідно. Це допомагає максимізувати ефективність та забезпечити досягнення поставлених цілей.

Враховуючи ці особливості, планування рекламної кампанії стає стратегічним процесом, спрямованим на досягнення максимальної ефективності та впливу на цільову аудиторію.

1.2 Особливості функціонування сучасних систем планування рекламної кампанії

Існує незліченна кількість деталей — графіки, результати, ресурси, бюджети та робочі процеси — які використовуються для проведення маркетингових кампаній. Поєднання цих деталей привертає увагу кампанії, яка охоплює потрібних людей у потрібний час.

Ефективний інструмент керування кампанією має об’єднувати робочі процеси та збирати інформацію в одному місці. Це повинно дати можливість бачити в реальному часі роботу та прогрес у всіх функціях маркетингової організації.

Розглянемо кілька прикладів найпопулярніших систем планування рекламної кампанії, а саме: Airtable, Asana та Smartsheet.

Airtable — це підключена платформа програм, яка дозволяє користувачам створювати та налаштовувати робочі процеси на основі спільних даних. Серед маркетингових команд одним із найпопулярніших випадків його використання є

керування кампаніями. Це тому, що гнучкість Airtable дозволяє легко підключати та узгоджувати календарі вмісту, плани запуску, творчі результати тощо [2].

На відміну від вузькоспеціалізованих точкових рішень, розроблених для виконання одного завдання, Airtable дозволяє великим відділам і командам співпрацювати та узгоджувати робочі процеси. Airtable — це гнучке, всеохоплююче рішення, яке об’єднує команди та забезпечує співпрацю в реальному часі.

Особливості, які виділяють Airtable:

1. Інтерактивна реляційна база даних: Airtable синхронізує інформацію в режимі реального часу між командами. Наприклад, коли член команди змінює дату запуску в Airtable, ця нова дата з’являється для кожного учасника команди, якого це стосується, у кожному пов’язаному робочому процесі.
2. Агрегування наборів даних: використовуючи Airtable можливо агрегувати всі набори даних — ефективність, часові рамки, бюджети, ресурси, дії, аудиторії та канали — для покращення всебічної видимості та планування кампанії.
3. Динамічні перегляди: Airtable дає змогу ділитися відповідними календарями та майбутніми програмами з зацікавленими сторонами каналу (такими як продажі та платні ЗМІ), щоб стимулювати ефективне просування та максимізувати вплив кампанії.
4. Масштабування до 250 000 записів і вдосконалена автоматизація: Airtable створено для організацій корпоративного рівня, обробляючи до 100 000 елементів на таблицю та 250 000 елементів на базу з нашим планом Enterprise. Завдяки автоматизації великі команди можуть перетворити свою базу на машину, масштабуючи операції за допомогою дій на основі тригерів.

Asana — це платформа спільного управління проектами, яку маркетингові команди використовують для організації та візуалізації проектів, планування спринтів і досягнення цілей. Управління кампанією є одним із найпопулярніших випадків його використання, оскільки воно допомагає полегшити міжкомандне спілкування та підзвітність.

Оскільки Asana не є реляційною базою даних, вона не синхронізує інформацію між проектами, а також існує ризик дублювання інформації в кількох робочих процесах, тому важко зрозуміти, яка версія є найточнішою. Тим не менш, це ефективний, простий у використанні інструмент для роботи з невеликими командами над дедлайнами та звітування про статус кампанії [3].

Особливості Asana:

Автоматизація завдань: Asana дозволяє встановлювати правила для автоматизації повторюваних і виснажливих завдань. Це чудово підходить для керування спілкуванням команди, публікаціями в соціальних мережах і сповіщеннями про схвалення.

Моніторинг у режимі реального часу: Asana дозволяє керівництву бачити прогрес своєї команди в режимі реального часу — щойно проект або завдання оновлюється, це відображається на всій платформі. Це також дозволяє легко оцінити пропускну здатність команди, щоб визначити, хто може взятися за завдання.

Велика бібліотека шаблонів: Asana пропонує понад 140 безкоштовних шаблонів для різноманітних випадків використання, включаючи творче виробництво та управління проектами. Це спрощує початок роботи, тож ви, швидше за все, побачите швидке впровадження в маркетинговій організації.

Smartsheet — це інструмент для спільного керування роботою на основі електронних таблиць. За допомогою нього можливо призначати завдання

користувачам, відстежувати проекти, керувати календарями, ділитися документами тощо.

Незважаючи на те, що Smartsheet не оновлюється в режимі реального часу та не дозволяє користувачам зберігати власні перегляди, його розширені функції керування проектами роблять його надійним варіантом для команд, які хочуть організувати свої кампанії.

Найбільш практичні функції:

1. Автоматизація робочого процесу: Smartsheet пропонує кілька автоматизованих шаблонів для виконання завдань, таких як нагадування, ініційовані тригерами, переміщення даних, оновлення аркушів тощо.
2. Функції керування проектами: платформа пропонує залежності, візуалізацію критичного шляху, керування ресурсами тощо, що забезпечує ефективну організацію завдань і відстеження проекту.

Розглянувши дані приклади можна зробити висновок, що в кожної системи є певні особливості, які допомагають користувачам полегшити створення системи планування рекламної кампанії. В системі планування рекламної кампанії на основі аналізу вебтрафіку та поведінки користувачів матиме особливість, що не присутня в жодній вище розглянутій системі, а саме моніторинг інтересів та поведінкових особливостей користувачів.

1.3 Постановка задачі по створенню системи планування рекламної кампанії

Для створення системи планування рекламної кампанії на основі аналізу даних вебтрафіку та поведінки користувачів представлено такі складові:

- Збір та обробка даних: Задача полягає в розробці механізму для збору та обробки даних веб-трафіку та поведінки користувачів. Це може

включати інтеграцію з аналітичними інструментами, встановлення відповідних тегів та збір необхідних метрик для подальшого аналізу.

- Аналіз даних: Основна задача полягає в розробці алгоритмів та моделей, що дозволяють аналізувати зібрані дані. Це включає визначення тенденцій, ідентифікацію цільових аудиторій, виявлення патернів поведінки користувачів та визначення ефективних маркетингових стратегій.
- Персоналізація реклами: Задача полягає в розробці механізмів для персоналізації рекламних повідомлень та контенту на основі зібраних даних. Це може включати створення сегментів аудиторії, розробку алгоритмів рекомендацій та індивідуального налаштування рекламних кампаній для кожного користувача.
- Прогнозування та оцінка результатів: Задача полягає в розробці методів прогнозування та оцінки результатів рекламних кампаній на основі зібраних даних. Це допомагає компанії приймати обґрунтовані рішення щодо розподілу рекламного бюджету та вдосконалення стратегій маркетингу.
- Розробка зручного інтерфейсу користувача: Задача полягає в розробці інтуїтивно зрозумілого та зручного інтерфейсу користувача для керування та моніторингу рекламних кампаній, візуалізації аналітичних даних та прийняття рішень.

Загальна мета створення такої системи полягає у покращенні ефективності рекламних кампаній, збільшенні конверсій та продажів, а також зниженні витрат на маркетинг шляхом точного визначення цільових аудиторій.

РОЗДІЛ 2
ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ СИСТЕМИ ПЛАНУВАННЯ РЕКЛАМНОЇ КАМПАНІЇ

2.1 Архітектура рекомендаційної системи

Існує безліч різних архітектур інформаційних систем, оскільки вони можуть бути спеціалізованими для різних завдань та областей застосування, наприклад: централізована, ієрархічна, розподілена та клієнт-серверна архітектури.

Централізована архітектура — це модель, в якій вся система або її велика частина функціонує на одному центральному вузлі чи сервері. Цей центральний вузол відповідає за керування та обробку всіх запитань та операцій, пов'язаних з функціонуванням системи. В даній архітектурі один центральний вузол приймає рішення та керує всіма аспектами роботи системи. Це дозволяє взяти під контроль всі процеси та ресурси. Централізована архітектура полегшує управління системою, оскільки всі важливі рішення та контроль зосереджені в одному місці. В централізованої архітектури також є недоліки, такі як одна точка відмови, недостатня масштабованість та зменшена надійність.

Ієрархічна архітектура - це модель, в якій компоненти системи організовані у вигляді ієрархії або структури з певним рівнем підпорядкованості та взаємозв'язку [4]. У такій архітектурі елементи групуються на різних рівнях, де кожен рівень має свої власні обов'язки та функції. Компоненти такої системи організовані у вигляді логічних рівнів. Кожен рівень може виконувати конкретні функції або виділяти певні сервіси. Комунікація та взаємодія між компонентами відбувається переважно в рамках їхнього власного рівня або з рівнями, розташованими вище або нижче в ієрархії. Система може легше розширюватися

або модифікуватися, оскільки нові рівні можуть бути додані або існуючі рівні модифіковані.

Розподілена архітектура (або розподілена система) - це модель розробки програмного забезпечення, при якій компоненти системи розташовані на різних фізичних чи логічних вузлах та взаємодіють між собою через мережу. Основною ідеєю є те, що складна система може бути розділена на окремі частини, які можуть працювати незалежно і взаємодіяти між собою, щоб виконати загальні завдання. Компоненти системи з розподіленою архітектурою можуть бути розташовані на різних фізичних машинах або віртуальних середовищах, що може бути великою відстанню один від одного.

Дана система матиме клієнт-серверну архітектуру (рис. 2.1). Модель даної системи реалізується так, що клієнтська частина системи відправляє запит на сервер, де він обробляється, і відповідь сервера відправляється клієнтові. Сервер може обслуговувати кілька клієнтів одночасно. Коли одночасно приходить кілька запитів, вони стають в чергу і сервер виконує їх послідовно. Іноді запити можуть мати пріоритети. Запити з вищими пріоритетами мають виконуватися раніше [5].

Рисунок 2.1 – Клієнт-серверна архітектура

Функції, що реалізуються на сервері:

- доступ, зберігання, резервне копіювання і захист даних;
- обробка запиту клієнта;

– відправка результату (відповіді) клієнту.

Функції, що реалізуються на стороні клієнта:

– надання користувальницького інтерфейсу;
– створення запиту до сервера і відправка запитів;
– отримання результатів і відправка додаткових команд (запитів на додавання, оновлення або видалення даних).

2.2 Обґрунтування вибору технологій та інструментів для реалізації системи

ASP.NET Core (Active Server Pages .NET Core) – це один з найпопулярніших платформ розробки вебдодатків. Дана платформа надає широкі можливості для створення потужних, масштабованих та безпечних вебдодатків.

ASP.NET пропонує модель програмування на основі сервера, де обробка даних та бізнес-логіка відбуваються на серверному боці. Це забезпечує переваги в області керування станом, безпеки, швидкості та надійності. За допомогою ASP.NET можна створювати вебдодатки з використанням різних мов програмування, таких як C#, VB.NET, F# та інші. Незважаючи на те, що ASP.NET значною мірою добре підходить для структури речення з ASP, він також надає іншу модель програмування та структуру для більш універсальних і стабільних програм, яка забезпечує більшу безпеку. Ви можете без вагань розширити свої поточні програми ASP, поступово додаючи до них корисні властивості ASP.NET [6].

Ключова перевага ASP.NET - це висока продуктивність. ASP.NET використовує компіляцію на стороні сервера, що дозволяє генерувати високоефективний виконуваний код. Крім того, платформа підтримує різні методи оптимізації завантаження сторінок, кешування та масштабування, що

дозволяє забезпечити швидку та ефективну роботу вебдодатків, при великому навантаженні [7].

ASP.NET також надає функціональні можливості, що полегшують розробку вебдодатків. Він має зручні інструменти для роботи з базами даних, зокрема Entity Framework [8]. Також, платформа надає розширюваність через використання компонентів, таких як віджети, модулі та бібліотеки, що допомагають прискорити процес розробки та забезпечити більшу функціональність.

ASP.NET також має вбудовані засоби безпеки, що дозволяють захистити вебдодатки від різних загроз. Він надає можливості для авторизації, автентифікації, захисту від атак та контролю доступу. Це дозволяє забезпечити високий рівень безпеки вебдодатку.

Backend системи планування рекламної кампанії на основі аналізу даних вебтрафіку та поведінки користувачів буде реалізовано за допомогою ASP.NET Core з використанням мови програмування C#.

Система планування рекламної кампанії буде мати користувачів, розподілених по ролях, тому є необхідність використовувати бібліотеку ASP.NET Core Identity. ASP.NET Core Identity – це вбудована бібліотека для управління авторизацією, автентифікацією та управління користувачами в вебдодатках, розроблених на платформі ASP.NET Core [9]. Авторизація – це керування засобами та рівнями доступу до певних ресурсів, та ресурсів системи, залежно від пароля та ідентифікатора користувача. Автентифікація – процедура встановлення належності користувачеві інформації в системі пред’явленого ним ідентифікатора.

ASP.NET Core Identity надає достатні функціональні можливості для реалізації модуля ідентифікації, що дозволяє забезпечити обмеження доступу до різних ресурсів додатку та безпеку. Дана бібліотека базується на концепції

клеймів, ролей та політик безпеки, що забезпечує гнучке налаштування доступу для різних прав ролей користувачів. Вона підтримує автентифікацію з використанням різних методів, таких як пароль, зовнішні постачальники (такі як Facebook, Google), реєстрацію нових користувачів та дозволяє виконувати двофакторну автентифікацію для більшого рівня безпеки.

Основні компоненти ASP.NET Core Identity включають:

- Користувачі та ролі: Бібліотека надає можливість створювати, оновлювати та видаляти користувачів, а також надає зручний інтерфейс для роботи з ролями користувачів. Це дозволяє легко управляти доступом до різних функцій додатка.
- Аутентифікація та авторизація: ASP.NET Core Identity забезпечує механізми аутентифікації, що дозволяють перевіряти ідентичність користувачів. Крім того, вона надає можливість встановлювати рівень авторизації для кожної дії або ресурсу в додатку.
- Зберігання даних: Бібліотека підтримує різні провайдери для зберігання даних, такі як SQL Server, SQLite, MySQL та інші. Це дає можливість вибрати потрібний провайдер залежно від потреб додатку.
- Зовнішні постачальники: ASP.NET Core Identity підтримує інтеграцію з різними зовнішніми постачальниками аутентифікації, такими як Facebook, Google, Twitter та інші. Це дозволяє користувачам використовувати свої облікові записи з цих сервісів для входу в додаток.

Загалом, ASP.NET Core Identity є потужним і зручним інструментом для розробки системи ідентифікації в веб-додатках на платформі ASP.NET Core. Вона спрощує реалізацію безпеки та керування доступом, дозволяючи розробникам швидко створювати надійні та безпечні додатки.

В додатку, що розробляється, використовується база даних для зберігання даних користувачів, їх поведінки тощо. Для зручної взаємодії з базою даних для додавання, редагування, виведення та видалення даних необхідно використовувати Entity Framework.

Entity Framework (EF) — це платформа об’єктно-реляційного відображення (ORM) з відкритим кодом для ADO.NET. Спочатку він постачався як невід’ємна частина .NET Framework, однак, починаючи з Entity Framework версії 6.0, він постачався окремо від .NET Framework.

Entity Framework — це набір технологій в ADO.NET, який підтримує розробку програмних програм, орієнтованих на дані. Архітекторам і розробникам програм, орієнтованих на дані, зазвичай важко досягти двох дуже різних цілей. Вони повинні моделювати сутності, зв’язки та логіку бізнес-проблем, які вони вирішують, а також працювати з механізмами обробки даних, які використовуються для зберігання та отримання даних [10]. Дані можуть охоплювати кілька систем зберігання, кожна зі своїми власними протоколами; навіть програми, які працюють з єдиною системою зберігання, повинні збалансувати вимоги системи зберігання з вимогами написання ефективного та підтримуваного програмного коду. Цю проблему зазвичай називають «невідповідністю об’єктно-реляційного імпедансу».

Багато інструментів об’єктно-реляційного відображення (ORM) (так званих «об’єктно-реляційних менеджерів») було розроблено, щоб дозволити розробникам працювати з даними у формі об’єктів і властивостей домену, таких як клієнти та адреси клієнтів, не турбуючись самі з базовими таблицями та стовпцями бази даних, де зберігаються ці дані. За допомогою ORM розробники можуть працювати на вищому рівні абстракції, коли вони мають справу з даними, а також можуть створювати та підтримувати програми, орієнтовані на дані, з

меншим кодом, ніж у традиційних програмах. Entity Framework — це рішення ORM, яке зараз просувається для використання в стеку розробки Microsoft.

Він надає достатні інструменти для роботи з базами даних, спрощуючи доступ, розробку і керування даними в програмах, написаних на платформі .NET.

Основні переваги Entity Framework включають:

- Об'єктно-орієнтований підхід: Entity Framework дозволяє розробникам працювати з базами даних у зручному об'єктно-орієнтованому стилі. Він дозволяє використовувати класи та об'єкти, що представляють таблиці та записи бази даних, замість прямої роботи з SQL-запитами [11]. Це полегшує розробку та підтримку коду.
- Автоматична генерація SQL-запитів: Entity Framework забезпечує автоматичну генерацію SQL-запитів на основі використання об'єктів та запитів LINQ (Language Integrated Query). Це дозволяє розробникам писати запити до бази даних мовою C# або Visual Basic, замість прямої роботи з SQL-мовою. Це спрощує роботу з базою даних та зменшує кількість ручного написання SQL-запитів.
- Міграції бази даних: Entity Framework надає можливість використовувати міграції для збереження та автоматичного оновлення схеми бази даних зі змінами в моделі даних. Це дозволяє зручно зберігати та відстежувати зміни в базі даних під час розвитку додатка, не втрачаючи дані.
- Підтримка різних провайдерів баз даних: Entity Framework підтримує різні провайдери баз даних, такі як SQL Server, MySQL, PostgreSQL, Oracle та багато інших. Це дозволяє розробникам працювати з різними системами управління базами даних без зміни коду додатка.

- Кешування та оптимізація запитів: Entity Framework надає можливості для кешування результатів запитів, що поліпшує продуктивність додатка. Він також включає механізми оптимізації запитів, що дозволяють виконувати ефективні запити до бази даних.
- Інтеграція з ASP.NET та іншими компонентами .NET: Entity Framework інтегрований з платформою .NET, зокрема з ASP.NET. Він надає можливості для швидкої розробки веб-додатків з доступом до бази даних. Крім того, він підтримує взаємодію з іншими компонентами .NET, такими як LINQ, WCF, WPF та інші.

Система планування рекламної кампанії на основі аналізу даних вебтрафіку та поведінки користувачів передбачає керування великими об’ємами даних, що зберігатимуться в базі даних. Для керування даними обрано СУБД MySQL.

MySQL - це відкрита система управління базами даних (СУБД), яка використовується для зберігання і управління даними у веб-додатках та інших програмах. Вона є однією з найпопулярніших реляційних СУБД у світі і використовується в багатьох веб-проектах і додатках[12].

Основні переваги MySQL включають:

- Висока продуктивність: MySQL відомий своєю швидкодією та високою продуктивністю. Він може ефективно обробляти великі обсяги даних та виконувати запити швидко [13], що робить його популярним в сферах, де швидкість доступу до даних є критичним фактором.
- Надійність та стабільність: MySQL є дуже стабільною та надійною СУБД. Він витримує великі навантаження та має вбудовані механізми забезпечення цілісності даних та відновлення після збоїв.
- Простота використання: MySQL має просту у використанні та зрозумілу мову запитів SQL, що робить його легким у навчанні та

розумінні. Це сприяє швидкому розгортанню та розробці баз даних [14].

- Гнучкість та розширюваність: MySQL підтримує широкий спектр функцій та можливостей. Він має можливості для виконання різних типів запитів, реплікації, кластеризації та розширення за допомогою різноманітних розширень та модулів.
- Сумісність: MySQL є сумісним з багатьма платформами та операційними системами, включаючи Windows, Linux та macOS. Він також підтримує різні мови програмування, такі як PHP, C#, Python, Java та інші, що дозволяє розробникам використовувати їх у своїх проектах.
- Активна спільнота: MySQL має велику та активну спільноту розробників, яка надає підтримку, документацію, розв'язання проблем та навчальні матеріали. Це робить MySQL відкритим для співпраці та обміну знаннями.

MySQL використовується у багатьох вебдодатках, корпоративних системах та проектах з великими обсягами даних. Він є потужним та надійним рішенням для зберігання та управління базами даних, що задовольняє потреби у роботі з даними.

Система планування рекламної кампанії на основі аналізу даних вебтрафіку та поведінки користувачів передбачає створення користувацького інтерфейсу для зручної взаємодіє клієнтів з додатком. Користувацький інтерфейс реалізовуватиметься на основі фреймворку Angular.

Angular — це платформа та фреймворк для створення односторінкових клієнтських програм за допомогою HTML і TypeScript. Angular написаний на TypeScript [15]. Він реалізує основні та додаткові функції як набір бібліотек TypeScript, які ви імпортуєте у свої програми.

Архітектура програми Angular спирається на певні фундаментальні концепції. Основними будівельними блоками фреймворку Angular є компоненти Angular, які організовані в NgModules. NgModules збирають пов’язаний код у функціональні набори; додаток Angular визначається набором NgModules. Додаток завжди має принаймні кореневий модуль, який уможливлює початкове завантаження, і зазвичай має набагато більше модулів функцій.

Angular надає багато переваг, зокрема:

- Компонентна архітектура: Angular побудований на основі компонентної архітектури, що дозволяє розбити додаток на незалежні компоненти. Компоненти можна повторно використовувати та легко управляти ними, що спрощує розробку та підтримку додатків.
- Прив'язка даних: Angular надає систему прив'язки даних, яка дозволяє автоматично оновлювати вміст сторінки при зміні даних в програмі [16]. Це дозволяє створювати динамічні та реактивні інтерфейси користувача.
- Розширення HTML: Angular використовує HTML з розширеними можливостями, такими як директиви, шаблони, пайпи та інші. Це дозволяє розробникам створювати більш потужні та гнучкі інтерфейси користувача.
- Управління станом: Angular надає механізми для ефективного управління станом додатка. Наприклад, він має вбудовану систему стану (Angular State Management) або може інтегруватись зі сторонніми бібліотеками керування станом, такими як NgRx. Це дозволяє ефективно керувати складним станом додатка та забезпечувати передбачувані зміни стану.

- Маршрутизація: Angular надає можливості маршрутизації, що дозволяє створювати різні сторінки та навігацію в додатку. Він дозволяє організувати структуру додатку та навігацію між його різними компонентами.
- Підтримка розширень: Angular має велику кількість сторонніх модулів, бібліотек та інструментів, які полегшують розробку та розширення додатків. Наприклад, існують бібліотеки для роботи з формами, анімацією, взаємодією з сервером, тестуванням та багато інших.

Angular є одним з найпопулярніших фреймворків для розробки веб-додатків, завдяки своїм потужним функціональним можливостям, великій спільноті розробників та підтримці від Google [17]. Він дозволяє створювати швидкі, масштабовані та сучасні додатки з високою продуктивністю та багатими функціональними можливостями.

Visual Studio Code (VS Code) є одним з найпопулярніших та потужних редакторів коду, розроблених компанією Microsoft. Цей безкоштовний, відкритий інструмент надає розширені можливості для розробки програмного забезпечення на різних платформах, включаючи Windows, macOS та Linux. VS Code зарекомендував себе як одне з найулюбленіших серед розробників завдяки своїй широкій функціональності, налаштуванням та розширюваності.

Перш за все, VS Code пропонує зручний та інтуїтивно зрозумілий користувацький інтерфейс, що дозволяє розробникам працювати з комфортом і ефективністю. Завдяки його легкому ваговому дизайну, редактор запускається швидко і має низьке споживання ресурсів, що робить його ідеальним вибором для проектів будь-якої складності.

Одна з ключових переваг VS Code - це його розширюваність. Редактор надає доступ до широкого спектру розширень, що дозволяють розробникам

налаштовувати його під свої потреби. За допомогою цих розширень можна додавати нові функціональні можливості, мови програмування, сніппети, теми оформлення та інші інструменти, що полегшують розробку. Крім того, VS Code підтримує інтеграцію з популярними системами керування версіями, такими як Git, що спрощує роботу в команді та ведення проектів [18].

VS Code також надає багато інструментів для розробки та налагодження програмного забезпечення. Він підтримує розширення для відлагодження, надає доступ до консолі, інтегрованого терміналу та інших корисних інструментів для розробників. Крім того, VS Code пропонує вбудовану підтримку для систем автоматизації завдань, таких як Gulp, Grunt та npm scripts, що полегшує процес розробки і збірки проєктів.

Завдяки своїм багатим функціоналом, розширюваністю та зручному інтерфейсу, Visual Studio Code став популярним вибором серед розробників. Він дозволяє прискорити та полегшити процес розробки програмного забезпечення, забезпечуючи високу продуктивність та зручність в роботі. Незалежно від платформи чи мови програмування, VS Code стане надійним партнером у розробці проєкту.

2.3 Моделювання системи планування рекламної кампанії на основі аналізу даних вебтрафіку та поведінки користувачів

Мета системи планування рекламної кампанії на основі аналізу вебтрафіку та поведінки користувачів - це оптимізувати та підвищити ефективність рекламних заходів, спрямованих на веб-аудиторію. Для досягнення цієї мети ставляться наступні цілі:

1. Аналіз поведінки користувачів: Збір та аналіз даних про поведінку користувачів на веб-сайті з метою зрозуміти їхні потреби, інтереси та попередній досвід.

2. Визначення цільової аудиторії: Встановлення якісних та кількісних характеристик цільової аудиторії на основі аналізу даних вебтрафіку, щоб краще спрямовувати рекламу.
3. Підвищення конверсії: Використання даних щодо поведінки користувачів для покращення конверсії в цільових діях (наприклад, покупок або реєстрацій) на веб-сайті.
4. Персоналізація контенту: Вдосконалення веб-контенту на основі даних про користувачів для забезпечення більш особистого та відповідного досвіду.
5. Оптимізація рекламних бюджетів: Використання аналітики для ефективного розподілу рекламного бюджету на різні канали та платформи, що найкраще взаємодіють з цільовою аудиторією.
6. Покращення рекламної стратегії: Аналіз результатів попередніх рекламних кампаній для розробки більш точної та ефективної стратегії на основі даних вебтрафіку.
7. Збільшення здатності прогнозування: Використання аналізу даних для прогнозування поведінки користувачів та реакції на рекламні заходи у майбутньому.
8. Моніторинг та відстеження показників: Створення системи моніторингу та відстеження ключових показників ефективності (KPI) для постійного оцінювання успіху рекламних кампаній.
9. Збільшення прибутковості: Покращення ефективності рекламних кампаній, щоб збільшити прибутковість бізнесу через залучення більше клієнтів та оптимізацію витрат.
10. Забезпечення конкурентної переваги: Використання аналізу вебтрафіку та поведінки користувачів для створення конкурентної переваги на ринку.

11. Забезпечення відповідності стандартам та регуляціям: Використання системи для забезпечення відповідності всім законодавчим вимогам та стандартам щодо обробки даних користувачів.

Мета та цілі є загальними для системи планування рекламної кампанії на основі аналізу вебтрафіку та поведінки користувачів і можуть змінюватися в залежності від конкретних потреб бізнесу та ринкових умов.

Система планування рекламної кампанії на основі аналізу даних вебтрафіку та поведінки користувачів має містити такі функціональні можливості:

- Збір даних користувачів з різних ресурсів. Система збиратиме дані користувачів, такі як демографічні дані, інтереси та хобі, поведінка покупців, лояльність, поведінка на соціальних медіа, локалізація, відгуки та рейтинги.
- Сегментація користувачів. Всі користувачі будуть розділені на певні групи для побудови статистики.
- Призначення відповідності певних сегментів користувачів певним рекламним оголошенням. Для більш продуктивної рекламної кампанії для окремих сегментів користувачів необхідно підбирати різні рекламні оголошення, що би відповідали їхнім уподобанням.
- Створення профілів користувачів. Для об’єднання даних окремих користувачів з різних ресурсів використовується номер телефону та електронна пошта, як ідентифікатори окремого користувача.
- Підбір реклами. Система підбиратиме рекламні рекомендації для користувачів різних ресурсів.
- Моніторинг результатів. Система виводитиме статистичні дані, що базуються на пропозиціях реклами на які користувачі відреагували позитивно, негативно або ж нейтрально.

Ці функціональні можливості забезпечать оптимізований підхід до планування рекламної кампанії, так як рекламні оголошення будуть підбиратись для всіх користувачів окремо на основі їх уподобань.

Аналіз вимог для системи планування рекламної кампанії на основі аналізу даних вебтрафіку та поведінки користувачів допоможе чітко визначити, які функції та можливості повинна мати система.

Система повинна збирати дані з різних джерел вебтрафіку, включаючи серверні журнали, файли журналів, дані зі сторонніх аналітичних сервісів тощо та інтегрувати ці дані в єдину систему для подальшого аналізу.

Система має забезпечити можливість аналізу дій користувачів на веб-сайті, включаючи перегляд сторінок, кліки, час перебування та інші дії. Реалізувати аналіз паттернів поведінки та сегментацію користувачів на основі їхньої активності.

Система має дозволяти автоматично адаптувати контент веб-сайту або рекламу на основі індивідуальних інтересів користувачів.

Відповідно до описаних вище функцій, дана система буде взаємодіяти з різними ресурсами та рекламними менеджерами. Взаємодія з різними ресурсами буде реалізована за допомогою API (рис. 2.2).

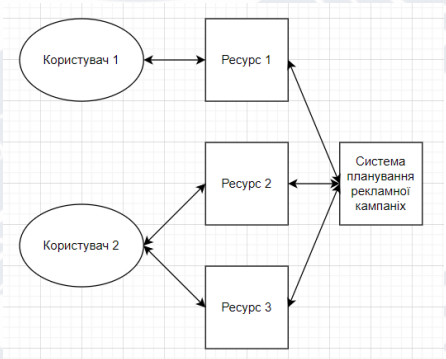

Рисунок 2.2 – Схема взаємодії системи планування рекламної кампанії з ресурсами

Ресурси відправлятимуть даній системі дані користувачів, їхні дії на ресурсах, що допоможе визначити їхні уподобання. Ресурси також можуть робити запити на рекламні оголошення. Дані рекламні оголошення будуть орієнтовані на кожного користувача індивідуально.

Для менеджерів система буде доступна у формі вебдодатоку. Це означає, що менеджери напряму звертатимуться до системи без ресурсів-посередників (рис. 2.3).

У версії вебдодатку система матиме дві частини: frontend та backend. Менеджери будуть взаємодіяти з користувацьким інтерфейсом, з якого відправлятимуться API запити на серверну частину системи. Тіло запитів буде представлено у JSON форматі.

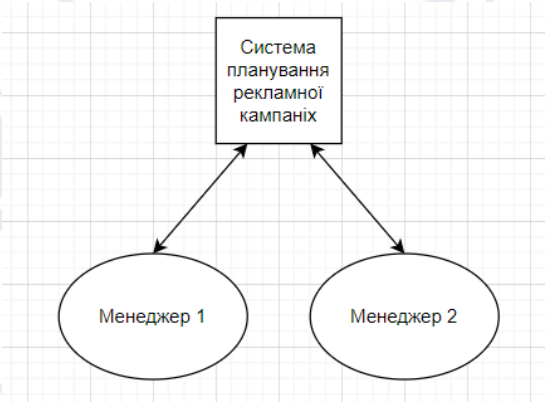

Рисунок 2.3 – Схема взаємодії системи планування рекламної кампанії з менеджерами

Система планування рекламної кампанії матиме декілька частин, а саме frontend, backend і база даних. Запити з різних ресурсів приходитимуть в backend частину. Менеджери взаємодіятимуть з системою через користувацький інтерфейс (frontend), з якого відправлятимуться запити на серверну частину.

Серверна частина буде складатись з програмної оболонки, написаної за допомогою ASP.NET Core та бази даних.

Як видно з концептуальної моделі (рис. 2.4), база даних (БД) складатиметься з десяти таблиць: Resources, Customers, Managers, Events, Preferences, Advertising, RestCustEvent, AdvertPrefs, PrefEvents та CustAdvertReactions. Три з них (RestCustEvent, AdvertPrefs, PrefEvents) є допоміжними таблицями для реалізації зв’язку багато до багатьох.

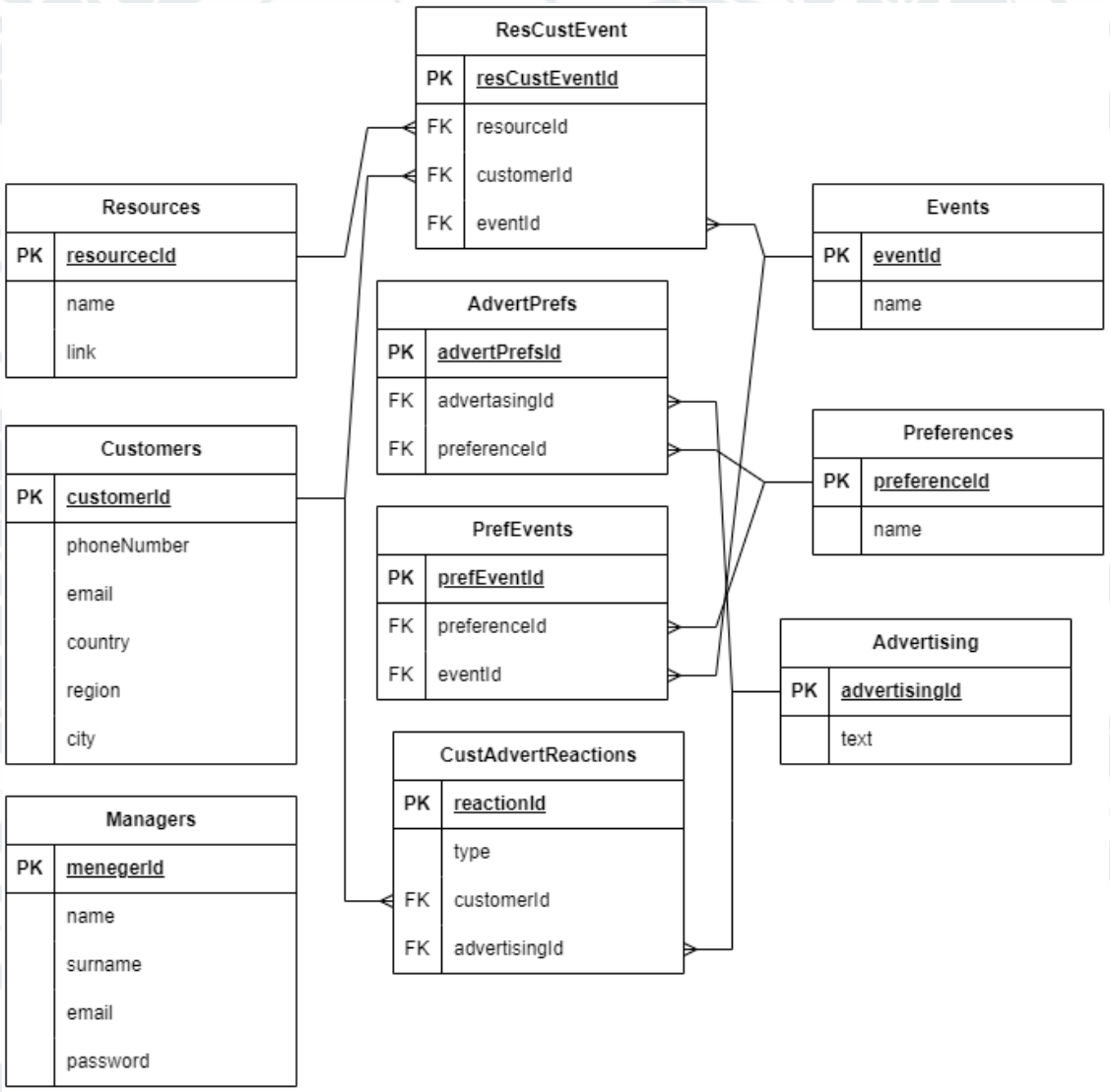

Рисунок 2.4 – Концептуальна діаграма бази даних

Дана модель має третю нормальну форму та відповідає наступним вимогам:

- Таблиці залежать винятково від первинного ключа;
- Уникнення повторень груп (категорії даних, що можуть зустрічатись різну кількість разів в різних записах) правильно визначаючи неключові атрибути;
- Атомарність: кожен атрибут має мати лише одне значення, а не множину значень;
- Дані, які з'являються повторно в декількох рядках, виносяться в окремі таблиці;
- Будь-яке поле, яке залежить від первинного ключа та від будь-якого іншого поля, має виноситись в окрему таблицю.

В моделі первинні ключі представлені як PK (primary key). Первинний ключ - це унікальне поле або набір полів в таблиці, яке ідентифікує кожен запис (рядок) в цій таблиці унікальним чином. Головна функція первинного ключа полягає в забезпеченні унікальності ідентифікаційних значень в таблиці та встановленні зв'язку між записами в таблиці і їхніми відношеннями до інших таблиць (якщо вони існують в БД) [19].

Основні характеристики первинного ключа:

1. Унікальність: Кожне значення в полі первинного ключа має бути унікальним для всіх записів в таблиці. Це допомагає однозначно ідентифікувати кожен запис [20].
2. Обов'язковість: Поле первинного ключа не може мати значення NULL (порожній). Кожен запис має мати значення в цьому полі.
3. Спрощення пошуку та з'єднань: Первинний ключ допомагає ефективно виконувати операції сортування, пошуку та з'єднання

записів в таблиці, оскільки він забезпечує унікальну адресацію кожного запису [21].

4. Використання для встановлення відносин: Поле первинного ключа може бути використане для створення відносин між таблицями.

Прикладом може служити таблиця "Managers" в БД. У цій таблиці поле "managerId" визначене як первинний ключ. Це гарантує, що для кожного менеджера буде унікальне ідентифікаційне значення, і його managerId може використовуватися для встановлення зв'язків з іншими таблицями, наприклад, з таблицею "Events", де може бути зовнішній ключ, що посилається на поле managerId.

Вторинний ключ (або зовнішній ключ) представлений як FK (foreign key). Вторинний ключ, також відомий як індекс, в БД - це поле або набір полів у таблиці, яке використовується для покращення швидкості та ефективності операцій пошуку та сортування даних. Відмінність від первинного ключа полягає в тому, що вторинний ключ не обов'язково є унікальним. Одне і те ж значення вторинного ключа може зустрічатися в кількох записах таблиці.

Основні характеристики вторинного ключа:

1. Не обов'язкова унікальність: Значення вторинного ключа не обов'язково повинні бути унікальними. Але в багатьох випадках вони можуть бути унікальними, проте це не є обов'язковою умовою.
2. Покращення швидкості операцій: Вторинний ключ допомагає покращити швидкість операцій сортування та пошуку, так як БД може використовувати індекси, щоб ефективно локалізувати необхідні записи.
3. Використання в запитах: Вторинні ключі зазвичай використовуються в SQL-запитах для фільтрації, відбору і сортування даних.

4. Зовнішні ключі: Вторинні ключі можуть бути використані як зовнішні ключі в інших таблицях для встановлення відносин між таблицями.
5. Вимагають певних ресурсів: Побудова та підтримка індексів (вторинних ключів) може потребувати додаткових обчислювальних та зберігальних ресурсів.

Вторинні ключі допомагають значно підвищити продуктивність БД, особливо при роботі з великими обсягами даних, оскільки вони дозволяють швидше знаходити та отримувати необхідну інформацію.

Таблиця Managers призначена для збереження даних рекламних менеджерів, які можуть користуватись даною системою та містить наступні поля:

- managerId – унікальний ідентифікатор менеджера;
- name – ім’я менеджера;
- surname – прізвище менеджера;
- email – електронна адреса менеджера;
- password – пароль для входу в клієнтську частину системи.

Таблиця Customers призначена для збереження даних користувачів різних ресурсів, яким пропонуватиметься реклама та на уподобання яких орієнтуються рекламні менеджери ведучи рекламну кампанію. Дана таблиця містить поля:

- customerId – унікальний ідентифікатор користувача;
- phoneNumber – номер телефону користувача;
- email – електронна адреса користувача;
- country – країна, в якій живе користувач;
- region – регіон, в якому перебуває користувач;
- city – місто, в якому перебуває користувач.

Таблиця Customers має два зв’язки: один до багатьох з таблицею CustAdvertReactions та багато до багатьох з таблицями Resources і Eventes.

Зв'язок "один до багатьох" (One-to-Many) є одним із типів відносин в БД, який вказує на те, що одному запису в одній таблиці відповідає багато записів в іншій таблиці. Цей тип відношення досить поширений в БД і використовується для моделювання багатьох реальних взаємозв'язків. Зв'язок "один до багатьох" є важливим для організації даних та моделювання складних відносин між об'єктами в БД. Він дозволяє ефективно зберігати та обробляти дані, які мають структуру "один до багатьох", і є одним із основних принципів реляційної БД.

Зв'язок "багато до багатьох" (Many-to-Many) в контексті БД вказує на те, що кожен запис в одній таблиці може бути пов'язаний з кількома записами в іншій таблиці, і навпаки, кожен запис в другій таблиці може бути пов'язаний з кількома записами в першій таблиці. Цей тип зв'язку виникає, коли існує складна багато-до-багатьох взаємодія між двома сутностями, і обидві сутності можуть бути пов'язані одна з одною через іншу таблицю, яку часто називають таблицею-посередником або таблицею зв'язків.

Таблиця Resources призначена для збереження даних про ресурси та містить поля:

- resourceId – унікальний ідентифікатор ресурсу;
- name – найменування ресурсу;
- link – посилання на ресурс.

Таблиця Resources має один зв’язок багато до багатьох з таблицями Customers та Events, що реалізується за допомогою таблиці-посередника RestCustEvent.

Таблиця Events зберігає дані про події, які користувач може виконати на певних ресурсах та містить поля:

- eventId – унікальний ідентифікатор події;
- name – найменування події.

Таблиця Events має два зв’язки багато до багатьох з таблицями Resources, Customers та з таблицею Preferences.

Таблиця Preferences призначена для збереження типів уподобань користувачів та містить поля:

- preferenceId – унікальний ідентифікатор уподобання;
- name – найменування уподобання.

Таблиця Preferences має два зв’язки багато до багатьох з таблицями Events та Advertising.

Таблиця CustAdvertReactions призначена для збереження даних про реакції користувачів на певні рекламні оголошення і має наступні поля:

- reactionId – унікальний ідентифікатор реакції;
- type – тип реакції, наприклад, перейдено за посиланням та закрито;
- customerId – зовнішній ключ, що вказує який саме користувач відреагував на рекламне оголошення;
- advertisingId – зовнішній ключ, що вказує на яку саме рекламу відреагував користувач.

Дана таблиця має два зв’язки багато до одного з таблицями Customers та Advertising.

Таблиці ResCustEvent, AdvertPrefs та PrefEvents є таблицями зв’язків та поєднують таблиці Resources, Customers, Events, Advertising та Preferences.

Таблиці зв'язків (або таблиці-посередники) - це спеціальні таблиці, що використовуються для реалізації зв'язків багато-до-багатьох між двома або більше таблицями. Ці таблиці дозволяють встановлювати зв'язок між записами в різних таблицях, де кожен запис може бути пов'язаним з багатьма записами в іншій таблиці, і навпаки. Таблиці зв'язків допомагають вирішити проблеми, пов'язані з багато-до-багатьох взаємодією в БД, і вони є важливою частиною

проектування БД для забезпечення правильного зберігання та доступу до даних при таких взаємозв'язках.

Серверна частина системи поділятиметься на компоненти: Data Access Layer (DAL), Business Logic Layer (BLL), View Models (VM), Controllers. Поділ на дані компоненти дозволяє покращити розширюваність та загалом продуктивність роботи системи.

Принцип роботи серверної частини (рис. 2.5):

1. клієнт робить запит до серверної частини у вигляді JSON формату;
2. система приймає запит та відправляє його до відповідного контролера;
3. контролер перевіряє правильність введених даних;
4. контролер передає запит до BLL;
5. BLL перевіряє, чи можливо здійснити даний запит;
6. BLL відправляє запит до DAL;
7. DAL приймає запит і опрацьовує його взаємодіючи з БД;
8. DAL відправляє (не обов’язково) відповідь BLL;
9. BLL відправляє (не обов’язково) відповідь до контролера;
10. контролер формує відповідь та відправляє клієнту.

"Data Access Layer" (DAL) - це рівень (шар або компонент) в архітектурі програмного забезпечення (ПЗ), який відповідає за доступ до даних і взаємодію з БД [22]. Цей рівень має роль посередника між бізнес-логікою додатку (рівень BLL - Business Logic Layer) та джерелами даних, такими як БД, файли, зовнішні сервіси і т. д.

Data Access Layer є частиною архітектури програми, що відповідає за взаємодію із зберіганням даних, таким як база даних. В .NET Core, для роботи з базами даних використовуються об'єкти Entity Framework Core (EF Core).

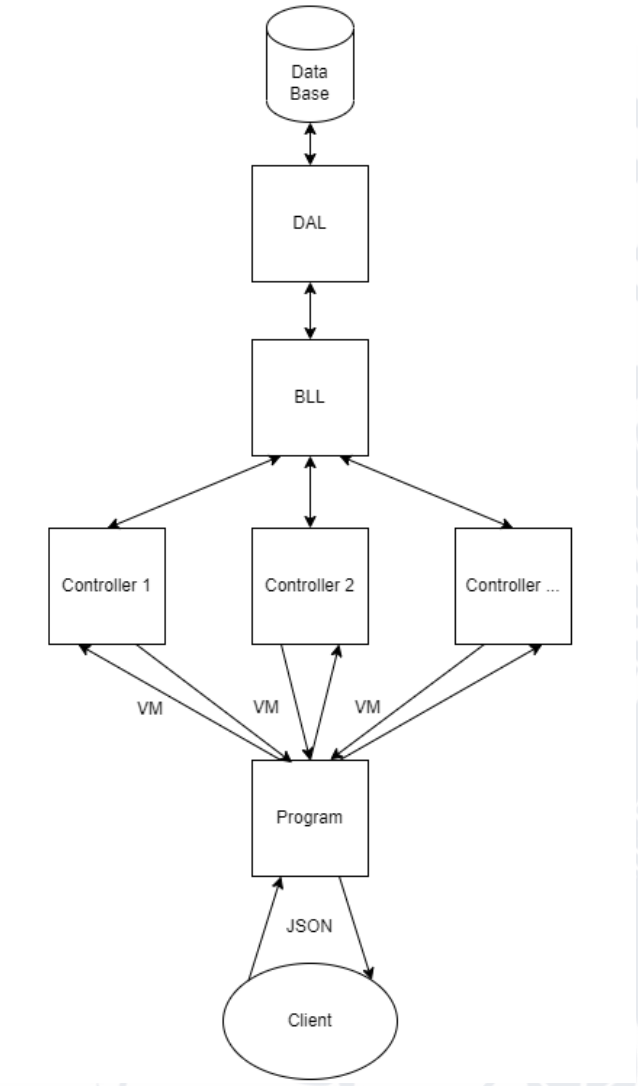

Рисунок 2.5 – Схема взаємодії компонентів серверної частини системи

Основні завдання та функції DAL включають:

– DAL встановлює підключення до БД і виконує операції для вставки, отримання, видалення та оновлення даних в БД;
– DAL відповідає за створення і виконання SQL-запитів або викликів для взаємодії з БД. Це включає в себе створення SQL-запитів, параметризацію запитів та обробку результатів;

- у деяких випадках, для спрощення взаємодії з БД, DAL може використовувати ORM-фреймворки, такі як Entity Framework, NHibernate або Dapper. ORM дозволяє робити роботу з БД більш об'єктно-орієнтованою і приховувати деталі SQL-запитів;
- DAL може забезпечувати кешування даних для підвищення швидкості доступу до них та зниження навантаження на БД;
- DAL може виконувати управління підключеннями до БД, включаючи відкриття, закриття та переходи в режим очікування;
- DAL може забезпечувати підтримку транзакцій для групового виконання операцій з БД з можливістю відкату у разі виникнення помилок;
- DAL повинен мати механізми для обробки та відстеження помилок, які можуть виникнути під час взаємодії з БД.

Data Access Layer розділяє бізнес-логіку додатку від деталей роботи з БД, що полегшує розробку, тестування і зміну додатку. Він є важливою складовою архітектури додатку, особливо в веб-додатках, де доступ до даних є важливою частиною функціональності [23].

Business Logic Layer (BLL) або шар бізнес-логіки - це рівень або компонент в архітектурі програмного забезпечення, який містить логіку та правила, пов'язані з бізнес-процесами і обробкою даних в додатку [24]. Цей рівень відповідає за виконання операцій, які стосуються бізнес-логіки та виконання бізнес-правил.

Основні функції та завдання Business Logic Layer включають:

- валідація даних: BLL відповідає за перевірку і валідацію даних, які надходять в додаток. Це включає в себе перевірку на відповідність бізнес-правилам та обмеженням;

- виконання бізнес-операцій: BLL виконує операції, які визначають логіку додатку. Це може включати в себе обчислення, перетворення даних, розрахунки та інші операції, необхідні для виконання бізнес-процесів;
- взаємодія з Data Access Layer (DAL): щоб отримати або зберегти дані, BLL взаємодіє з DAL. Воно створює запити до бази даних для отримання або збереження інформації. Через цей рівень відбувається обмін даними з базою даних;
- управління транзакціями: бізнес-логіка може управляти транзакціями, особливо в разі потреби в атомарних операціях (діях, які виконуються або відкатуються повністю);
- реалізація бізнес-правил: BLL включає в себе логіку, яка визначає бізнес-правила та умови, за якими додаток працює. Це може включати в себе перевірки на доступ до ресурсів, авторизацію користувачів і інші важливі аспекти безпеки;
- обробка винятків: BLL відповідає за обробку винятків та помилок, які можуть виникнути під час виконання бізнес-логіки, і приймає рішення щодо їх обробки або передачі на вищий рівень для обробки помилок;
- забезпечення безпеки: в BLL можуть бути вбудовані механізми безпеки для забезпечення захисту даних та відповідності політикам безпеки додатку.

Business Logic Layer допомагає відокремити бізнес-логіку від інфраструктурних деталей та забезпечує більшу модульність, розширюваність і тестируваність додатку. Він є однією із ключових частин архітектури додатку та сприяє підтримці чистого та структурованого коду.

Розподіл (розділення) логіки на рівні BLL (Business Logic Layer) і DAL (Data Access Layer) в додатку ASP.NET Core має декілька важливих переваг і відіграє ключову роль в структурі та організації додатку [25]. Розподіл на рівні робиться для:

- розділення обов'язків: розділення логіки на рівні BLL і DAL допомагає дотримуватися принципу розділення обов'язків (Separation of Concerns). Кожен рівень відповідає за свої конкретні функції і завдання, що спрощує розробку, тестування та підтримку коду;
- модульність і розширюваність: розподіл логіки дозволяє створювати окремі модулі або класи для різних аспектів додатку, таких як бізнес-логіка та доступ до даних. Це полегшує додавання нових функцій і розширення додатку без впливу на інші частини коду;
- тестування: розділення допомагає покращити тестованість коду. Можна легко тестувати бізнес-логіку окремо від доступу до даних, використовуючи юніт-тести. Це дозволяє виявляти та виправляти помилки в бізнес-логіці незалежно від роботи з базою даних;
- підтримка різних джерел даних: іноді може виникнути потреба в підтримці різних джерел даних, таких як бази даних, зовнішні сервіси, файлова система і т. д. Розділення логіки дозволяє змінювати частини DAL без зміни BLL, і навпаки;
- більша чіткість коду: код стає більш читабельним і легким для розуміння, коли бізнес-логіка відокремлена від коду доступу до даних. Це сприяє командній роботі та знижує ризик виникнення помилок;
- забезпечення безпеки: розділення логіки дозволяє застосовувати правила безпеки, наприклад, обмеження доступу до бази даних, на рівні DAL, забезпечуючи вищу безпеку даних;
- можливість використання різних технологій: рівень DAL може бути адаптованим до використання різних технологій доступу до даних (в даному випадку Entity Framework), не змінюючи бізнес-логіку.

Загалом, розподіл на рівні BLL і DAL є загальною практикою програмування і допомагає зробити додаток більш організованим, підтримуваним і розширюваним.

Контролери (controllers) - це класи, які відповідають за обробку HTTP-запитів, які надходять системи. Контролери виконують централізовану обробку запитів і приймають рішення щодо того, які дії (actions) повинні бути виконані відповідно до отриманих запитів [26].

Основні характеристики контролерів:

- обробка запитів: контролери служать для обробки HTTP-запитів, включаючи GET, POST, PUT, DELETE та інші методи HTTP;
- визначення дій: внутрішні методи контролера, які називаються "діями" (actions), визначають, які конкретні дії повинні бути виконані відповідно до запиту. Дії відповідають на запити і виконують необхідні операції, такі як відображення сторінок, обробка даних, взаємодія з базою даних і інше;
- маршрутизація: контролери також визначають маршрутизацію запитів, яка вказує, які URL-шляхи (роути) відповідають кожній дії. Маршрутизація визначає, який контролер і яка дія повинні бути викликані при певному URL;
- параметри запиту: контролери можуть приймати параметри з запиту, такі як дані форми, рядки запиту, параметри маршруту і т. д. Ці параметри передаються в аргументах методів-дій контролера;
- фільтри дій: в ASP.NET Core, до контролерів можуть бути додані фільтри дій (action filters), які дозволяють впроваджувати логіку до і після виконання дій, наприклад, для автентифікації, авторизації, кешування, логування тощо;

– застосування сервісів і бізнес-логіки: контролери можуть взаємодіяти з іншими компонентами додатку, включаючи сервіси та бізнес-логіку, для виконання дій, пов'язаних з бізнес-процесами.

View models (моделі представлення) представляють собою класи, які використовуються для передачі даних з контролера (або бізнес-логіки) до представлення (шаблону вигляду) для відображення на сторінці веб-додатку. View models допомагають вирішити проблеми пов'язані з передачею і відображенням даних від бізнес-логіки до представлення і забезпечують кращу організацію та керованість даними в шаблонах вигляду. View models є важливим інструментом в ASP.NET додатках для управління даними, які відображаються на сторінках вигляду, і для забезпечення більшої контрольованості і розширюваності додатків.

Клієнтська частина реалізовуватиметься на платформі Angular та міститиме наступні частини: кореневий модуль, модулі, компоненти, сервіси, маршрутизація, моделі даних, ресурси та конфігурація (рис. 2.6).

Кореневий модуль (App Module), який ініціалізує Angular додаток та налаштовує основні параметри, такі як маршрутизація, HTTP-сервіси та інші. Модуль імпортується з @angular/core. Кореневий модуль є центральним модулем в Angular-додатку і відповідає за ініціалізацію та налаштування додатку. Цей модуль використовується для об'єднання різних функціональних модулів та компонентів в єдину структуру додатку.

Загалом, кореневий модуль є важливим елементом архітектури Angular-додатку і визначає головну структуру та конфігурацію додатку. Всі інші модулі та компоненти додатку можуть бути організовані навколо цього кореневого модуля.

У Angular, модулі (modules) використовуються для організації та групування функціональної частини додатку. Модулі сприяють покращенню

структури проекту, розділенню функцій і забезпеченню легкої розширюваності та підтримки коду. Модуль може містити користувацькі компоненти, директиви, піпи, сервіси, інтерсептори і будь-які інші Angular-ресурси. Це дозволяє організувати код додатку у логічні розділи.

Angular дозволяє завантажувати модулі ліниво (lazy-loading). Це означає, що модуль буде завантажено тільки в тому випадку, якщо користувач перейде на сторінку, яка вимагає компоненти з цього модуля. Це полегшує завантаження додатку та покращує швидкість завантаження сторінок.

Модулі використовуються для організації маршрутів в Angular. Можна визначити, який модуль та компонент відображатиметься при певних URL-шляхах. Маршрутизація допомагає відокремити функціональність додатку на різних сторінках.

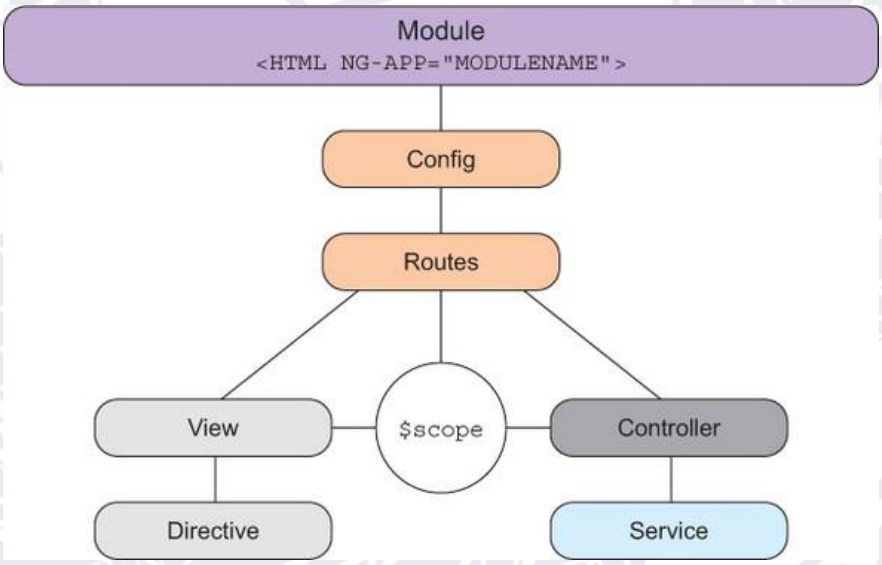

Рисунок 2.6 – Структура додатку Angular

Загалом, модулі в Angular є потужним інструментом для організації коду додатку, забезпечують його модульність та підтримують кращу структуру

проекту. Вони дозволяють створювати розширюваний та легко підтримуваний код для великих та складних додатків [27].

Компоненти є однією з основних будівельних одиниць в Angular і відповідають за відображення та контроль користувацького інтерфейсу в додатку. Кожен компонент Angular має власний клас TypeScript, шаблон HTML і CSS стилі, що роблять його незалежним та переносимим. Компоненти є ключовими для створення динамічних та інтерактивних додатків в Angular. Вони роблять можливим розподіл функціональності на невеликі та самостійні частини, що полегшує розробку та обслуговування програмного коду.

Сервіси (services) в Angular - це одна з фундаментальних концепцій, які допомагають організовувати та розподіляти логіку бізнес-логіки і функціональність, яка не пов'язана з відображенням, між компонентами та модулями системи. Сервіси відокремлюють код, який відповідає за обробку даних, взаємодію з сервером, маніпуляцію даними та іншу бізнес-логіку від компонентів, які відображають дані на сторінці. Це сприяє підтримці принципу однієї відповідальності (Single Responsibility Principle) та полегшує тестування.

Сервіси в Angular відіграють важливу роль у структурі та функціональності додатку, допомагаючи забезпечити модульність, повторне використання коду та відокремлення бізнес-логіки від інтерфейсу користувача.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ПЛАНУВАННЯ РЕКЛАМНОЇ КАМПАНІЇ НА ОСНОВІ АНАЛІЗУ ДАНИХ ВЕБТРАФІКУ ТА ПОВЕДІНКИ КОРИСТУВАЧІВ

3.1 Вимоги до системи планування рекламної кампанії

Створення рекомендаційної системи для планування рекламної кампанії на основі аналізу даних вебтрафіку та поведінки користувачів є завданням, яке вимагає ретельного підходу та врахування різноманітних аспектів. Нижче наведені ключові вимоги до такої системи: збір та збереження даних, аналіз та обробка даних, персоналізація рекомендацій, інтеграція з рекламними платформами, оптимізація бюджету та конфіденційності, моніторинг та звітність, масштабованість.

Збір та збереження даних являє собою здатність збирати, обробляти та зберігати великі обсяги даних вебтрафіку та поведінки користувачів. Рекомендаційна система має забезпечувати ефективне зберігання даних для подальшого аналізу та генерації рекомендацій.

Аналіз та обробка даних представляє можливість використання алгоритмів аналізу даних та статистичних методів для виявлення патернів, та трендів у вебтрафіку та поведінці користувачів, а також здатність розпізнавати ключові ознаки, такі як інтереси, попередні взаємодії та конверсії.

Персоналізація рекомендацій – це розробка алгоритмів, які забезпечують персоналізовані рекомендації для кожного користувача на основі його унікальних характеристик та інтересів, здатність враховувати контекст та змінювати рекомендації в реальному часі.

Інтеграція з рекламними платформами являє собою можливість інтегрувати рекомендаційну систему з різними рекламними платформами, щоб

автоматично налаштовувати рекламні кампанії, підтримку стандартів рекламних API для забезпечення сумісності та ефективної інтеграції.

Оптимізація бюджету та результативності представляє необхідність реалізації механізму для визначення оптимального розподілу рекламного бюджету з урахуванням ефективності кампаній та конверсій, автоматичні стратегії оптимізації для підвищення ефективності рекламної діяльності.

Забезпечення безпеки та конфіденційності – це механізми шифрування та захисту даних користувачів. Відповідність нормативам щодо конфіденційності та захисту особистих даних, таких як GDPR.

Моніторинг та звітність являє собою інструменти для моніторингу ефективності рекомендаційної системи та рекламних кампаній, а також забезпечення можливості генерації звітів та аналізу результатів для прийняття обґрунтованих рішень.

Врахування цих вимог допоможе побудувати ефективну рекомендаційну систему, яка сприятиме успішному плануванню та реалізації рекламних кампаній на основі аналізу вебтрафіку та поведінки користувачів.

3.2 Програмна реалізація

База даних створена за допомогою інструменту міграцій в Entity Framework. Міграції (Migrations) в Entity Framework (EF) - це механізм, який дозволяє автоматично оновлювати схему бази даних при змінах в моделі даних (Code-First підходу) або при ручному визначенні змін в базі даних (Code-First або Database-First підходи). Міграції в EF допомагають управляти версіями бази даних та зберігати ці зміни в контролі версій, щоб розробники могли легко відстежувати та впроваджувати зміни в різних середовищах.

Створення бази даних розподілилось на такі кроки:

1. створення моделей даних;

2. додавання до моделей полів, призначених для реалізації зв’язків між таблицями;
3. створення контексту бази даних;
4. конфігурація зв’язків між моделями;
5. запуск міграції.

Створено десять моделей сутностей: Advertising, AdvertPref, CustAdvertReaction, Customer, Event, Preference, PrefEvent, Resource, RestCustEvent та User. Всі моделі і міграції розміщені в модулі DAL, так як даний модуль відповідає за безпосередній зв’язок з базою даних.

Модель Customer (рис. 3.1) являє собою запис з таблиці Customers. Дана модель містить всі поля таблиці, а саме: CustomerId, PhoneNumber, Email, Country, Region та City. Ця модель також містить поля для забезпечення зв’язку з моделями RestCustEvent та CustAdvertReaction. Так як обидва зв’язки мають тип один до багатьох, це означає, що один користувач може мати багато реакцій та багато моделей RestCustEvent. Тому дані поля є колекціями.

```
namespace appadd.DAL.Models;

public class Customer
{
    public long CustomerId { get; set; }
    public int PhoneNumber { get; set; }
    public string Email { get; set; }
    public string Country { get; set; }
    public string Region { get; set; }
    public string City { get; set; }

    public ICollection<RestCustEvent> RestCustEvents { get;
set; }

    public ICollection<CustAdvertReaction> CustAdvertReactions
{ get; set; }
}
```

Рисунок 3.1 – Лістинг моделі Customer

Модель Advertising (рис. 3.2) являє собою запис з таблиці Advertisings. Дана модель містить всі поля таблиці, а саме: AdvertisingId та Text. Ця модель також містить поля для забезпечення зв’язку з моделями Preference та CustAdvertReaction. Так як зв’язок з CustAdvertReaction є зв’язком один до багатьох, це означає, що одне рекламне оголошення може мати багато реакцій, тому дане поле є колекцією.

```
namespace appadd.DAL.Models;
public class Advertising
{
    public long AdvertisingId { get; set; }
    public string Text { get; set; }

    public List<AdvertPref> AdvertPrefs { get; set; }
    public ICollection<Preference> Preferences { get; set; }

    public ICollection<CustAdvertReaction> CustAdvertReactions
{ get; set; }
}
```

Рисунок 3.2 – Лістинг моделі Advertising

Для забезпечення зв’язку багато до багатьох через модель AdvertPref створено два додаткових поля: AdvertPrefs та Preferences. Поле AdvertPrefs є списком та містить записи таблиці AdvertPrefs. Поле Preferences є колекцією та містить усі записи Preference, з якими поєднане дане рекламне оголошення.

Модель CustAdvertReaction (рис. 3.3) являє собою запис з таблиці CustAdvertReactions. Дана модель містить всі поля таблиці, а саме: ReactionId та Type. Властивість Type має тип ReactionTypes та може мати значення, одне з варіантів: Ignore, Follow або Closed.

Модель CustAdvertReaction також містить поля для забезпечення зв’язку з моделями Customer та Advertising. Так як між цими моделями зв’язок один до

багатьох, одна реакція може мати одну рекламну пропозиція та одного користувача, який відреагував на неї.

```
namespace appadd.DAL.Models;
public enum ReactionTypes
{
    Ignore = 0,
    Follew = 1,
    Closed = 2
}

public class CustAdvertReaction
{
    public long ReactionId { get; set; }
    public ReactionTypes Type { get; set; }

    public long CustomerId { get; set; }
    public Customer Customer { get; set; }
    public long AdvertisingId { get; set; }
    public Advertising Add { get; set; }
}
```

Рисунок 3.3 – Лістинг моделі CustAdvertReaction та списоку ReactionTypes

Модель Event (рис. 3.4) являє собою запис з таблиці Events. Дана модель містить всі поля таблиці, а саме: EventId та Name. Дана модель також містить поля CustomerId, Customer, AdvertisingId та Add для забезпечення зв’язку з моделями RestCustEvent та Preferences. Так як між моделями Event та RestCustEvent зв’язок один до багатьох, однин Event може мати кілька моделей RestCustEvent.

Для забезпечення зв’язку багато до багатьох через модель PrefEvent створено два додаткових поля: PrefEvents та Preferences. Поле PrefEvents є списком та містить записи таблиці PrefEvents. Поле Preferences є колекцією та містить усі записи Preference, з якими поєднане дана подія.

```
namespace appadd.DAL.Models;

public class Event
{
    public long EventId { get; set; }

    public string Name { get; set; }
    public ICollection<RestCustEvent> RestCustEvents { get;
set; }

    public List<PrefEvent> PrefEvents { get; set; }
    public ICollection<Preference> Preferences { get; set; }
}
```

Рисунок 3.4 – Лістинг моделі Event

Модель Preference (рис. 3.5) являє собою запис з таблиці Preferences. Дана модель містить всі поля таблиці, а саме: EventId, Name та Link. Дана модель також містить поля для забезпечення зв’язку з моделлю RestCustEvent. Так як між моделями Resource та RestCustEvent зв’язок один до багатьох, однин Resource може мати кілька моделей RestCustEvent, тому поле RestCustEvents є колекцією.

```
namespace appadd.DAL.Models;
public class Resource
{
    public long ResourceId { get; set; }
    public string Name { get; set; }
    public string Link { get; set; }
    public ICollection<RestCustEvent> RestCustEvents { get;
set; }
}
```

Рисунок 3.5 – Лістинг моделі Resource

Для управління підключенням до бази даних створено контекст, що являє собою клас ApplicationContext (рис. 3.6). Контекст бази даних (DbContext) в Entity Framework - це важлива частина ORM (Object-Relational Mapping) підходу і

представляє собою об'єкт, який відповідає за взаємодію між додатком і базою даних.

```
namespace appadd.DAL;

public class ApplicationContext : IdentityDbContext<User>
{
    public DbSet<Resource> Resources { get; set; }
    public DbSet<Advertising> Advertisings { get; set; }
    …

    public
ApplicationContext(DbContextOptions<ApplicationContext>
options)
        : base(options) { }

    protected override void
OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        …
    }

    protected override void OnModelCreating(ModelBuilder
modelBuilder)
    {
        ……
        base.OnModelCreating(modelBuilder);

        modelBuilder.ApplyConfiguration(new
RoleConfiguration());
    }
}
```

Рисунок 3.6 – Лістинг класу ApplicationContext

Клас ApplicationContext містить поля типу DbSet: Resources, Advertisings, Reactions, Customers, Events, Preferences та RestCustEvents. Дані поля призначені для збереження даних таблиць БД.

Даний клас містить метод OnModelCreating, в якому налаштовуються зв’язки між моделями даних (рис. 3.7).

```
modelBuilder.Entity<Preference>()
          .HasMany(p => p.Events)
              .WithMany(p => p.Preferences)
              .UsingEntity<PrefEvent>(
                  j => j
                       .HasOne(pt => pt.Event)
                       .WithMany(t => t.PrefEvents)
                       .HasForeignKey(pt => pt.EventId),
                  j => j
                       .HasOne(pt => pt.Preference)
                       .WithMany(p => p.PrefEvents)
                       .HasForeignKey(pt => pt.PreferenceId),
                  j =>
                  {
                      j.HasKey(t => new { t.EventId,
t.PreferenceId });
                  });
```

Рисунок 3.7 – Лістинг – налаштування зв’язку багато до багатьох між моделями Preference та Event

Також даний клас містить метод OnConfiguring. Цей метод використовується для налаштування параметрів підключення до бази даних, коли контекст бази даних створюється. Тобто, можна визначити параметри підключення безпосередньо в коді, якщо вони не були визначені раніше через конструктор контексту або іншим способом.

Після виклику кількох команд entity framework з консолі, автоматично створюються міграції, що є інструкцією для створення таблиць бази даних та формуються SQL запити до БД, які конфігурують БД.

Для безпосереднього зв’язку з базою даних було створено репозиторії. При виклику методів репозиторіїв система формує запит для бази даних і повертає відповідь.

Для кожного репозиторію створено інтерфейси, які реалізовують репозиторії. За допомогою інтерфейсів репозиторіїв можна легко замінювати

реалізацію цих репозиторіїв без змінення бізнес-логіки. Наприклад, можливо почати з реалізації репозиторіїв на основі Entity Framework, а потім перейти на іншу технологію (наприклад, Dapper) без змін в інших частинах коду, якщо реалізація репозиторіїв відповідає одному і тому ж інтерфейсу. Інтерфейси репозиторіїв допомагають зменшити залежності між класами системе. Класи бізнес-логіки можуть взаємодіяти лише з інтерфейсами репозиторіїв, і не потребують знання про конкретну реалізацію цих репозиторіїв.

Загалом, використання інтерфейсів для репозиторіїв є доброю практикою програмування, оскільки воно сприяє покращенню структури коду, робить код більш тестовим і забезпечує гнучкість та легкість обслуговування.

Інтерфейс репозиторію IResourceRepository (рис. 3.8) вказує на реалізацію базових методів додавання, перегляду, оновлення та видалення об’єктів Resource та властивість Count, що визначає кількість об’єктів в базі даних.

```
using appadd.DAL.Models;
namespace appadd.DAL.Repositories.Interfaces;
public interface IResourceRepository
{
    long Count { get; }
    void Add(Resource resource);
    void Update(Resource resource);
    void Remove(Resource resource);
    void Remove(long id);
    Resource GetResource(long id);

    IEnumerable<Resource> Get();
}
```

Рисунок 3.8 – Лістинг – інтерфейс репозиторію IResourceRepository

Клас репозиторію ResourceRepository (рис. 3.9) успадковує інтерфейс IResourceRepository та реалізує всі методи і властивості даного інтрефейсу. В

конструкторі репозиторій приймає контекст бази даних типу ApplicationContext, з набори даних якого використовуються в методах репозиторію.

```
public class ResourceRepository : IResourceRepository
{
    public long Count { get => _context.Resources.Count(); }
    private readonly ApplicationContext _context;

    public ResourceRepository(ApplicationContext context){
        _context = context;
    }

    public void Add(Resource resource) {
        _context.Resources.AddAsync(resource);
        _context.SaveChanges();
    }

    public void Remove(Resource resource) {
        _context.Resources.Remove(resource);
    }

    public void Remove(long id){
        Resource res = GetResource(id);
        Remove(res);
    }
    public void Update(Resource resource){
        _context.Resources.Update(resource);
        _context.SaveChanges();
    }
    public Resource GetResource(long id){
        return _context.Resources.First(r => r.ResourceId ==
id);
    }
    public IEnumerable<Resource> Get(){
        return _context.Resources;
    }
}
```

Рисунок 3.9 – Лістинг – клас репозиторію ResourceRepository

За такою аналогією було створено репозиторії AdvertisingRepository, CustomerRepository, EventRepository, PreferenceRepository, ReactionRepository та

RestCustEventRepository, які успадковують відповідні інтерфейси IAdvertisingRepository, ICustomerRepository, IEventRepository, IPreferenceRepository, IReactionRepository та IRestCustEventRepository. Після створення, всі репозиторії реєструються у файлі Program.cs (рис. 3.10).

```
…
var builder = WebApplication.CreateBuilder(args);
#region Adding repositories
builder.Services.AddScoped<IAdvertisingRepository,
AdvertisingRepository>();
builder.Services.AddScoped<ICustomerRepository,
CustomerRepository>();
builder.Services.AddScoped<IEventRepository,
EventRepository>();
builder.Services.AddScoped<IPreferenceRepository,
PreferenceRepository>();
builder.Services.AddScoped<IReactionRepository,
ReactionRepository>();
builder.Services.AddScoped<IResCustEventRepository,
RestCustEventRepository>();
builder.Services.AddScoped<IResourceRepository,
ResourceRepository>();
#endregion
…
```

Рисунок 3.10 – Лістинг – реєстрація репозиторіїв у файлі Program.cs

Репозиторії допомагають відокремити бізнес-логіку системи від логіки доступу до даних. Це важливо для забезпечення принципу однієї відповідальності (Single Responsibility Principle), де кожен клас відповідає лише за одну конкретну функцію. Репозиторії визначають контракт для доступу до даних, який не залежить від конкретної реалізації бази даних. Це дозволяє легко змінювати або розширювати джерело даних без необхідності змінювати бізнес-логіку. Завдяки репозиторіям можна легко змінювати або підміняти реалізацію доступу до даних. Наприклад, можна використовувати Entity Framework для роботи з SQL Server, а потім перейти на Dapper або іншу ORM або NoSQL

рішення без необхідності змінювати весь код бізнес-логіки. Репозиторії допомагають зберегти інкапсуляцію деталей доступу до даних. Класи бізнес-логіки взаємодіють лише з репозиторіями, не маючи прямого доступу до деталей бази даних, що полегшує обслуговування та розвиток системи.

Загалом, репозиторії є важливим архітектурним елементом для структурування та підтримки додатків ASP.NET та інших додатків, і вони допомагають забезпечити високий рівень абстракції та флексібільності в роботі з даними.

Сервіси допомагають відокремити бізнес-логіку системи від інфраструктурних аспектів, таких як обробка HTTP-запитів або доступ до бази даних. Це підтримує принцип єдиної відповідальності (Single Responsibility Principle), де кожен клас відповідає за одну конкретну функцію. Сервіси використовують репозиторії та інші механізми доступу до даних, і це допомагає абстрагувати бізнес-логіку від деталей роботи з базою даних. Це спрощує зміну джерела даних без впливу на логіку бізнесу.

Було додано інтерфейс IResourceService (рис. 3.11) для інкапсуляції методів доступу до даних репозиторіїв. Використання інтерфейсів для сервісів дозволяє легко замінювати реалізацію сервісів, не змінюючи код бізнес-логіки.

```
public interface IResourceService
{
    IEnumerable<Resource> List();

    void Add(Resource res);

    void Update(Resource res);

    void Remove(Resource res);

}
```

Рисунок 3.11 – Лістинг – інтерфейс сервісу IResourceService

Всі методи інтерфейсу IResourceService реалізовані в класі RepositoryService (рис. 3.12). Даний клас приймає об’єкти типу IResourceRepository в конструкторі, через який взаємодіє з моделями типу Resource. Клас реалізовує методи List, Add, Update, Remove.

```
public class ResourceService : IResourceService
{
    private IResourceRepository _resourceRopository;
    public ResourceService(IResourceRepository
resourceRopository){
        _resourceRopository = resourceRopository;
    }

    public void Add(Resource res){
        _resourceRopository.Add(res);
    }

    public IEnumerable<Resource> List() {
        return _resourceRopository.Get();
    }

    public void Remove(Resource res){
        _resourceRopository.Remove(res);
    }

    public void Update(Resource res){
        _resourceRopository.Update(res);
    }
}
```

Рисунок 3. 12 – Лістинг – клас ResourceService

По аналогії інтерфейсу IResourceService були створені IAdvertisingService, ICustomerService, IEventService, IPreferenceService, IReactionService та IRestCustEventService та класи, що їх успадковують, а саме: AdvertisingService, CustomerService, EventService, PreferenceService, ReactionService та RestCustEventService. Після створення, всі сервіси реєструються у файлі Program.cs (рис. 3.13).

```
var builder = WebApplication.CreateBuilder(args);
…
#region Adding services
builder.Services.AddTransient<IAdvertisingService,
AdvertisingService>();
builder.Services.AddTransient<ICustomerService,
CustomerService>();
builder.Services.AddTransient<IEventService, EventService>();
builder.Services.AddTransient<IPreferenceService,
PreferenceService>();
builder.Services.AddTransient<IReactionService,
ReactionService>();
builder.Services.AddTransient<IResCustEventService,
ResCustEventService>();
builder.Services.AddTransient<IResourceService,
ResourceService>();
#endregion
…
```

Рисунок 3.13 – Лістинг – реєстрація сервісів у файлі Program.cs

Загалом, створення сервісів є практикою програмування для розробки додатків, оскільки вони допомагають розділити функціональність, спрощують тестування, забезпечують легку заміну реалізації і полегшують підтримку та розвиток додатків.

В системі додано процедури авторизації та автентифікації. Якщо користувач не авторизований в системі, він переадресовується на сторінку авторизації (рис. 3.14). На сторінці авторизації користувач може авторизуватись ввівши свою пошту і пароль, або перейти на сторінку реєстрації.

Ці процедури реалізовані з використанням JWT ключів. JWT ключ (JSON Web Token ключ) - це секретний ключ, який використовується для підпису та перевірки JWT токенів. JWT - це стандартний спосіб вираження інформації між двома сторонами у вигляді JSON об'єкта. Цей токен зазвичай використовується для автентифікації і авторизації користувачів в додатках. JWT складається з трьох

частин: заголовка (Header), частини з корисною інформацією (Payload), і підпису (Signature). Підпис генерується за допомогою секретного ключа та даних з заголовка і корисної інформації, і використовується для перевірки цілісності токена.

Заголовок (Header) містить метадані токена та тип підпису, який використовується для підпису токена. Зазвичай, це JSON-об'єкт, який вказує тип токена (typ) і алгоритм підпису (alg). Тіло (Payload) - це місце, де розміщуються претензії (claims), тобто інформація, яку токен містить. Претензії можуть бути стандартними, такими як ідентифікатор користувача (sub), термін дії токена (exp) або користувацькі, створені для конкретного додатка. Підпис (Signature) додається до заголовку та тіла для створення підпису. Це забезпечує перевірку автентичності та цілісності токена.

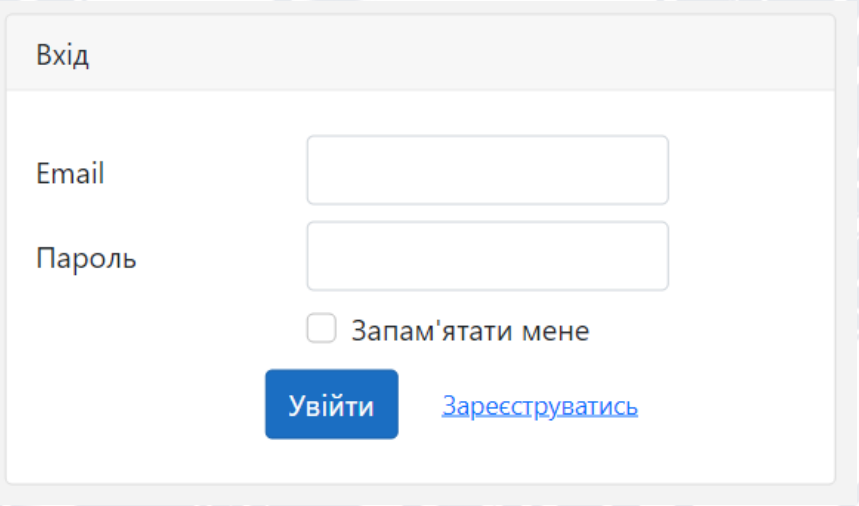

Рисунок 3.14 – Сторінка авторизації

Для проведення авторизації створено метод LoginAsync (рис. 3.15) в контролері AccountController. Клас AccountController приймає значення

userManager, JwtHandler та mapper в конструкторі та записує посилання на ці об’єкти у внутрішні приватні змінні.

```
[ApiController]
[Route("api/account")]
public class AccountController : ControllerBase
{

…

    [HttpPost]
    [Route("login")]
    public async Task<IActionResult> LoginAsync([FromBody]
LoginDTO model){
        var user = await
_userManager.FindByNameAsync(model.Email);
        if (user == null || !await
_userManager.CheckPasswordAsync(user, model.Password)){

            return BadRequest(new LoginResponseDTO {
ErrorMessage = "Неправильний логін або пароль" });
        }

        var signingCredentials =
_jwtHandler.GetSigningCredentials();
        var claims = await _jwtHandler.GetClaimsAsync(user);
        var tokenOptions =
_jwtHandler.GenerateTokenOptions(signingCredentials, claims);
        var token = new
JwtSecurityTokenHandler().WriteToken(tokenOptions);

        return Ok(new LoginResponseDTO { IsAuthSuccessful =
true, Token = token });
    }
}
```

Рисунок 3.15 – Лістинг – метод LoginAsync контролера AccountController

Метод LoginAsync є асинхронним. Даний метод приймає об’єкт model типу LoginDTO (рис. 3.16), який автоматично формується з тіла http запиту. В класі LoginDTO визначено дві властивості Email та Password. Метод перевіряє чи в базі

даних є запис про користувача з переданою електронною поштою, та чи вірний пароль для цього користувача. Якщо так, то формується jwt ключ, який передається у відповідь на запит клієнта, якщо ні – повертається помилка авторизації.

```
using System.ComponentModel.DataAnnotations;
namespace appadd.ViewModels.Account;
public class LoginDTO
{
    [Required(ErrorMessage = "Потрібно вказати email")]
    [EmailAddress(ErrorMessage = "Некоректний email")]
    public string Email { get; set; }

    [Required(ErrorMessage = "Потрібно вказати пароль")]
    public string Password { get; set; }
}
```

Рисунок 3.16 – Лістинг – LoginDTO

На стороні клієнта створено метод onSubmit (рис. 3.17), який викликається при натисканні на кнопку «Увійти».

Метод onSubmit формує запит з введених даних у форму та обробляє відповідь запиту. Якщо користувач натиснув на кнопку «Remember me», присланий jwt ключ записується в локальну пам’ять, якщо ні – в пам’ять сесії.

Користувацький інтерфейс реалізовано за допомогою Angular. Система має один основний модуль app.module, в якому конфігурована маршрутизація, декларовані компоненти, налаштовано головний компонент, підключено сервіси, конфігуровано jwt. Як видно з рисунку 3.18, система має чотири основні директиви: _interfaces, account, home, shared.

Директива _interfaces містить різні інтерфейси, що дозволяють визначати типи даних, що передаються запитами, типи даних що отримуються у відповідь від сервера та моделі збереження даних. Наприклад файл account.ts містить

інтерфейси: LoginRequest, LoginResponse, RegisterRequest та RegisterResponse. Всі ці інтерфейси відносяться до методів авторизації та реєстрації в системі.

```
onSubmit() {
    this.showError = false;
  …
    const user: LoginRequest = {
      email: this.loginForm.get(['email'])?.value,
      password: this.loginForm.get(['password'])?.value
    };
    this.logining = true;
    this.authService.loginUser('api/account/login', user)
    .subscribe({
      next: (res: LoginResponse) => {
        if (res.isAuthSuccessful) {
          if (this.loginForm.get(['rememberMe'])?.value !== '') {
            localStorage.removeItem('storage');
            localStorage.setItem('token', res.token);
          }
          else {
            localStorage.removeItem('token');
            localStorage.setItem('storage', 'session');
            sessionStorage.setItem('token', res.token);
          }

          this.router.navigateByUrl('/');
        }
        else {
          this.errorMessage = res.errorMessage;
          this.showError = true;
        }


this.authService.sendAuthStateChangeNotification(res.isAuthSuccessful);
      }
    });
  }
```

Рисунок 3.17 – Лістинг – метод авторизації клієнта

Директива account містить компоненти, в яких реалізуються функції реєстрації та авторизації, а саме: LoginComponent та RegisterComponent.

Директива home містить всі компоненти, в яких реалізуються функції панелі адміністратора, а саме перегляд, додавання, оновлення та видалення ресурсів, подій, уподобань, реклами та інших.

Директива shared містить класи, що можуть використовуватись по всій системі. До таких класів відносяться guards, кастомні валідатори та сервіси.

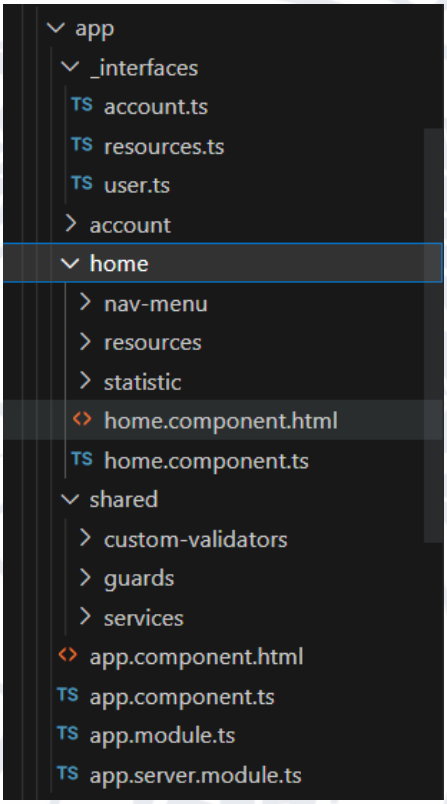

Рисунок 3.18 – Структура класів користувацького інтерфейсу

До guards відносяться класи, що роблять певні перевірки при переході користувачем по певному посиланню. Наприклад, клас AuthGuard (рис. 3.17) призначений для перевірки чи користувач авторизований при переході на всі сторінки панелі адміністратора.

Кастомні валідатори, що містяться в директиві shared, призначені для перевірки коректності введених користувачем даних, покращення UX, зменшення залежностей від шаблонів HTML, виправлення помилок перед відправленням на сервер та можливостей розширення функціональності та забезпечення безпеки системи.

```
@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {

  constructor(private authService: AuthenticationService, private
router: Router) {}

  canActivate(next: ActivatedRouteSnapshot, state:
RouterStateSnapshot) {
    if (this.authService.isUserAuthenticated()) {
      return true;
    }
    this.router.navigate([ '/account/login' ], { queryParams: {
returnUrl: state.url }});

    return false;
  }
}
```

Рисунок 3.19 – Лістинг – клас AuthGuard

В директиві shared також створено сервіси AuthenticationService, EnvironmentUrlService, ErrorHandlerService, UserService (рис. 3.20) та інші.

Сервіси в Angular використовуються для виконання різних завдань, пов'язаних із бізнес-логікою системи та обробкою даних. Сервіси типу UserService дозволяють розділити логіку доступу до даних від компонентів. Сервіси типу ResourcesService використовуються для збереження та обміну спільними даними між різними компонентами. Сервіс AuthenticationService використано для реалізації системи аутентифікації та авторизації користувачів. Він включає перевірку інформації про користувача, управління токенами та ролей, а також логіку доступу до захищених ресурсів. Сервіси дозволяють легко тестувати бізнес-логіку вашого додатку, оскільки ви можете створити одиниці тестування (unit tests) для сервісів та перевірити їхню правильність окремо від компонентів. Сервіси в Angular є важливими для створення розділеної та підтримуваної структури системи, а також для забезпечення перевикористання логіки та доступу до даних в різних частинах системи.

```
@Injectable({
  providedIn: 'root'
})
export class UserService {
  constructor(private http: HttpClient, private envUrl:
EnvironmentUrlService) { }

  public getEmail = (route: string) => {
    return this.http.get<UserResponse>
(this.createCompleteRoute(route, this.envUrl.urlAddress));
  }

  private createCompleteRoute = (route: string, envAddress:
string) => {
    return `${envAddress}/${route}`;
  }
}
```

Рисунок 3.20 – Лістинг – сервіс UserService

Для реалізації методу API для додавання відомостей про поведінку користувачів створено клас ResCustEventController (рис. 3.21).

```
[ApiController]
[Route("api/rescustevent")]
public class ResCustEventController : ControllerBase
{

    private IResCustEventService _resCustEventService;
    private readonly IMapper _mapper;

    public ResCustEventController(IResCustEventService
resCustEventService, IMapper mapper)
    {
        _resCustEventService = resCustEventService;
        _mapper = mapper;
    }
    ……
}
```

Рисунок 3.21 – Лістинг – Контролер ResCustEventController

Маршрут до методів класу ResCustEventController – api/rescustevent. Даний контролер приймає в конструкторі сервіс для роботи з даними, що визначають поведінку користувачів та мапер.

В класі ResCustEventController реалізовано метод Add (рис. 3.22) для додавання відомостей про поведінку користувачів. Метод приймає об’єкт класу AddResCustEventRequestDTO, що автоматично формується з тіла запиту.

```
[HttpPost]
[Route("add")]
public IActionResult Add([FromBody] AddResCustEventRequestDTO
model)
{
    if (model == null || !ModelState.IsValid)
    {
        var errors = new List<string>();

        foreach (ModelStateEntry modelState in
ModelState.Values) {
            foreach (ModelError error in modelState.Errors) {
                errors.Add(error.ErrorMessage);
            }
        }
         return BadRequest(new ResponseDTO { Errors = errors
});
    }

    RestCustEvent res = _mapper.Map<RestCustEvent>(model);
    _resCustEventService.Add(res);

    return StatusCode(201, new ResponseDTO { IsSuccessful =
true });
}
```

Рисунок 3.22 – Лістинг – метод Add класу ResCustEventController

Метод Add виконує перевірку чи дані об’єкту model є коректними та у разі знаходження некоректних даних повертає помилки. За допомогою мапера об’єкт model трансформується в об’єкт класу RestCustEvent. Викликавши метод Add

сервісу IResCustEventService, додається новий запис, який свідчить, що певний користувач зробив певну дію на певній платформі.

Для реалізації методу API для створення списку пропозицій реклами створено контролер AddController (рис. 3.23). Маршрут до методів класу ResCustEventController – api/add. Даний контролер приймає в конструкторі сервіси для роботи з рекламою, користувачами ресурсів та мапер.

```
[ApiController]
[Route("api/add")]
public class AddController : ControllerBase
{
    private IAdvertisingService _addService;
    private ICustomerService _custService;
    private readonly IMapper _mapper;

    public AddController(IAdvertisingService addService,
ICustomerService custService, IMapper mapper)
    {
        _addService = addService;
        _custService = custService;
        _mapper = mapper;
    }

    ……

}
```

Рисунок 3.23 – Лістинг – клас AddController

Метод приймає ідентифікатор користувача, для якого формується список реклами, що формується з рядка url. В клас AddController додано метод Get (рис. 3.24), для отримання рекомендацій по рекламі для певного користувача. Рекомендації підбираються шляхом зчитування дій користувача та його реакцій на рекламні оголошення, відповідно до яких формується список інтересів. Якщо список інтересів пустий, користувачу пропонується найпопулярніші пропозиції.

```
[HttpGet]
[Route("get")]
public IActionResult Get([FromQuery] long custId){
    Customer customer = _custService.Get(custId);
    if (customer != null)
        return BadRequest(new GetAddsResponseDTO {
            Errors = new List<string>() { "Non-existent
customer." }});
    var preferences = customer.RestCustEvents.SelectMany(rce
=> rce.Event.Preferences);
    var prefWeights = new Dictionary<long, int>();
    foreach (Preference pref in preferences) {
        if (prefWeights.ContainsKey(pref.PreferenceId))
            prefWeights[pref.PreferenceId] += 1;

        else
            prefWeights.Add(pref.PreferenceId, 1);
    }

    var adds = _addService.GetAdds(prefWeights);
    return StatusCode(200, new GetAddsResponseDTO {
        IsSuccessful = true,
        Adds = adds
    });
}
```

Рисунок 3.24 – Лістинг – метод Get контролера AddController

В методі Get контролера AddController спочатку виконується перевірка, чи існує даний користувач. Після цього формується список уподобань даного користувача, який передається в метод GetAdds сервісу IAdvertisingService. Метод GetAdds повертає список реклами для даного користувача, який відправляється у відповідь на запит.

3.3 Тестування та верифікація системи

Тестування та верифікація рекомендаційної системи — це процеси, спрямовані на перевірку та підтвердження правильності роботи системи, а також її відповідності вимогам та очікуванням.

Тестування рекомендаційної системи включає функціональне тестування, верифікацію точності та тестування стійкості.

Функціональне тестування — це тип тестування програмного забезпечення, спрямований на перевірку того, чи виконуються функціональні вимоги програми та чи працює вона відповідно до специфікацій та очікувань [28]. Перевірка, як система реагує на конкретні вхідні дані та як вона генерує рекомендації. Включає тестування основних функцій, таких як персоналізовані рекомендації, адаптація до змін у поведінці користувачів тощо. Основний акцент робиться на тому, як програма повинна працювати з точки зору функціональності.

Основні характеристики функціонального тестування включають:

- вхідні та вихідні дані: перевірка правильності обробки вхідних даних програмою та перевірка вірності та цілісності вихідних даних;
- функціональні операції: тестування окремих функцій програми для перевірки їх правильності та відповідності вимогам, визначення того, чи програма правильно взаємодіє з іншими компонентами системи;
- робочі процедури: перевірка правильності виконання робочих процедур та алгоритмів, тестування послідовності виконання операцій та їх вплив на систему;
- керування даними: перевірка правильності керування даними (додавання, зміна, видалення), тестування обробки базових операцій баз даних (якщо застосовується);
- умови використання: тестування програми в різних умовах використання, включаючи максимальне та мінімальне навантаження;
- помилки та виняткові ситуації: тестування програми на виявлення помилок та реакцію на непередбачувані ситуації, перевірка відновлення програмою після виникнення помилок;

Функціональне тестування спрямоване на те, щоб впевнитися в тому, що програма виконує ті функції, для яких вона була створена, і відповідає вимогам користувачів чи бізнес-замовників [29].

Верифікація точності в контексті рекомендаційних систем означає перевірку того, наскільки ефективно система передбачає вподобання та інтереси користувачів. Точність є ключовим критерієм оцінки ефективності рекомендаційної системи і вказує на те, наскільки вірно система може передбачити, які об'єкти зацікавлять конкретного користувача.

Етапи верифікації точності:

1. Створення тестового набору: формування набору тестових даних, які включають в себе вхідні дані (наприклад, історію перегляду, покупки, оцінки) та очікувані вихідні результати (рекомендації, які користувач може очікувати);
2. Застосування рекомендаційної системи до тестового набору: подача тестового набору даних до рекомендаційної системи з метою отримання передбачених рекомендацій;
3. Порівняння результатів: Порівняння отриманих рекомендацій з очікуваними результатами для визначення того, наскільки точно система передбачає вподобання користувачів;
4. Визначення метрик точності: використання різних метрик для кількісної оцінки точності, таких як середньоквадратична помилка, точність, відсоток покриття та інші;
5. Аналіз результатів та вдосконалення: аналіз отриманих метрик для визначення областей, де система може покращити свою точність, внесення корекцій у алгоритми та параметри моделей для оптимізації точності.

Цей процес повинен проводитися як в тестовому середовищі, так і в реальних умовах, враховуючи фактичні дані та поведінку користувачів. Верифікація точності є важливим етапом у розробці та оптимізації рекомендаційних систем для забезпечення їхньої ефективності та задоволення потреб користувачів.

Тестування стійкості (резистентності) — це вид тестування програмного забезпечення, спрямований на перевірку того, як система або програма веде себе під час дії в несприятливих чи екстремальних умов [30]. Цей вид тестування дозволяє визначити, наскільки стійкою є програма до помилок, неправильних вхідних даних, перевантаження, низької продуктивності, втрати зв'язку, атак і т.д.

Основні аспекти тестування стійкості включають:

1. Навантаження та продуктивність: визначення того, як програма працює під високим навантаженням, тестування продуктивності при надмірному обсязі даних чи великій кількості одночасних користувачів;
2. Відновлення після помилок: перевірка здатності системи або програми автоматично відновлюватися після виникнення помилок або аварій;
3. Стабільність: тестування, як система реагує на різні умови середовища, такі як зміни в ресурсах, мережеві відключення, перепади напруги тощо;
4. Тестування витривалості: визначення того, як програма веде себе під час тривалої роботи або витривалості навантаження;
5. Тестування сумісності: перевірка того, як програма веде себе в різних середовищах, на різних платформах чи при використанні різних версій операційних систем [31].

Тестування стійкості є важливою частиною в розробці програмного забезпечення, оскільки воно дозволяє виявити та усунути проблеми, пов'язані з невідомими чи несподіваними умовами експлуатації. Такі тести допомагають підвищити надійність та стабільність системи в реальних умовах експлуатації.

Верифікація рекомендаційної системи включає в себе валідацію моделей, порівняння з очікуваннями та тестування адаптації.

Валідація моделей - це процес перевірки та оцінки правильності та ефективності моделей, які використовуються у системі, на основі вхідних та вихідних даних [32]. У контексті рекомендаційних систем або інших систем машинного навчання, валідація моделей є ключовим етапом для переконання в їхній здатності адекватно вирішувати поставлені завдання та генерувати точні та надійні результати.

Основні кроки та аспекти валідації моделей включають:

- Розподіл даних: розділ вхідних даних на тренувальний і тестовий набори. Зазвичай, модель тренується на тренувальному наборі, а потім перевіряється на тестовому наборі, який не використовувався в процесі тренування;
- Тренування моделі: використання тренувального набору для навчання моделі. Підбір оптимальних параметрів та налаштувань моделі;
- Перевірка на тестовому наборі: використання тестового набору для оцінювання точності та ефективності моделі. Перевірка, наскільки добре модель узгоджується з реальними даними, які вона раніше не бачила;
- Крос-валідація: розширення валідації за рахунок використання крос-валідації, яка включає поділ вхідних даних на кілька піднаборів, тренування моделі на частині цих наборів та перевірку на іншій

частині цих наборів. Це дозволяє ефективніше використовувати доступні дані та отримувати більш стабільні результати;

- Метрики ефективності: визначення та вимірювання різних метрик ефективності, таких як точність, відновлення, F-міра та інші, в залежності від конкретного завдання;
- Визначення взаємодії з користувачами: оцінка впливу моделі на реальних користувачів або визначення того, як добре модель взаємодіє з реальними сценаріями використання [33].

Валідація моделей є важливою складовою в роботі з системами машинного навчання, оскільки вона допомагає гарантувати, що модель може ефективно вирішувати задачі та генерувати коректні та надійні результати в реальних умовах [34].

Порівняння з очікуваннями вказує на процес перевірки того, наскільки результати, отримані від системи, відповідають очікуванням користувачів чи стандартам, і як добре вони відображають бажану якість рекомендацій.

Ключові етапи порівняння з очікуваннями включають:

1. Формування очікувань: визначення того, що користувачі очікують від рекомендаційної системи. Це може включати в себе врахування історії перегляду, попередніх взаємодій, декларованих вподобань, рейтингів тощо;
2. Визначення критеріїв оцінки: створення метрик або критеріїв, за якими можна оцінити, наскільки рекомендації відповідають очікуванням. Це може бути точність передбачення, різноманіття рекомендацій, адаптація до зміни у веденні користувача тощо.
3. Застосування рекомендаційної системи: використання системи для отримання рекомендацій для конкретних користувачів чи сценаріїв;

4. Зіставлення результатів із зазначеними очікуваннями: порівняння отриманих рекомендацій із зазначеними очікуваннями для оцінки того, наскільки вони відповідають попереднім уявленням;
5. Аналіз різниці: визначення та аналіз відмінностей між отриманими рекомендаціями та очікуваннями. Це може включати в себе виявлення аномалій, помилок чи можливостей для покращення системи;
6. Внесення корекцій: в разі неспівпадання рекомендацій із зазначеними очікуваннями, внесення корекцій у моделі чи алгоритми системи для поліпшення результатів.

Порівняння з очікуваннями допомагає переконатися, що рекомендаційна система відповідає потребам користувачів та генерує значущі та зручні рекомендації відповідно до їхніх очікувань.

Тестування адаптації (Adaptation Testing) - це вид тестування програмного забезпечення, спрямований на перевірку здатності системи або програми адаптуватися до змін у своєму середовищі або умовах експлуатації [35]. Цей процес дозволяє визначити, наскільки ефективно програма може пристосовуватися до нових умов і продовжувати надавати коректні та стабільні результати.

Ключові аспекти тестування адаптації включають:

1. Зміна умов: провокування змін в середовищі чи умовах експлуатації, таких як зміни в мережі, обсязі даних, характеристиках користувачів тощо;
2. Автоматичне визначення змін: використання автоматизованих інструментів для визначення змін у середовищі або умовах, що може вплинути на систему;

3. Реакція системи: оцінка того, наскільки швидко і ефективно система реагує на зміни. Це може включати в себе перезавантаження, зміну конфігурацій, автоматичне налаштування тощо [36];
4. Оцінка стійкості: визначення того, наскільки стійкою є система під час адаптації до нових умов;
5. Тестування відновлення: перевірка того, як швидко та ефективно система може відновитися після змін чи викликаних проблем;
6. Тестування механізмів адаптації: оцінка ефективності та правильності вбудованих механізмів адаптації, таких як алгоритми автоматичного визначення конфігурацій, ресурсне управління тощо;
7. Взаємодія з іншими системами: тестування взаємодії з іншими системами під час змін в їхньому стані або конфігурації [37].

Тестування адаптації є важливим, особливо в умовах, коли програмне забезпечення повинне функціонувати в змінних чи непередбачуваних умовах. Цей вид тестування спрямований на гарантію того, що система може ефективно адаптуватися до різних сценаріїв використання та забезпечувати стабільну роботу в умовах зміни.

Загальний підхід до тестування та верифікації повинен бути систематичним, включати тестові набори, валідацію даних, тестування з реальними користувачами (якщо можливо) та інші методи, щоб забезпечити надійність та ефективність рекомендаційної системи.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи проаналізовано сучасні системи планування рекламної кампанії, розглянуто технології для створення рекомендаційної системи планування рекламної кампанії та створено рекомендаційну систему планування рекламної кампанії на основі аналізу даних вебтрафіку та поведінки користувачів.

Доведено актуальність розробки даної системи, поставлено задачу, а також підібрано та описано технології, необхідні для реалізації системи. Було описано основні переваги обраних технологій.

Для реалізації системи було підібрано клієнт-серверну архітектуру, що складається з бази даних, серверної та клієнтської частини. Реалізовано базу даних в СУБД MySQL, та додано основні сутності для збереження статистики поведінки користувачів на різних ресурсах.

На серверній частині реалізовано розподіл виконання запитів по рівнях контролерів, бізнес-логіки та рівню доступу до даних, а також описано як саме відбувається обробка запитів.

Створено користувацький інтерфейс, через який адміністратори можуть управляти ресурсами, подіями та пропозиціями. Було створено API, через які відбувається збір даних з ресурсів та передача рекламних пропозицій для окремих користувачів або для всіх користувачів ресурсу загалом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Таран Н.М., Зелінська О.В. Рекомендаційна система планування рекламної кампанії на основі аналізу даних вебтрафіку та поведінки користувачів. Матеріали IV Всеукраїнської науково-практичної конференції «Комп'ютерні технології обробки даних»: збірник наукових праць – Вінниця, ДонНУ імені Василя Стуса, 2023. С.179-181.
2. The 10 best campaign management tools for marketing teams. URL: https://www.airtable.com/articles/marketing/marketing-campaign-management-software (дата звернення 12.09.2023).
3. Що таке Asana project management — як спростити ваш робочий процес. URL: https://cloudfresh.com/ua/cloud-blog/shho-take-asana-project-management-yak-sprostyty-vash-robochyj-protses/ (дата звернення 12.09.2023).
4. Зелінська О.В., Матвійчук Р.Д. Переваги та недоліки ієрархічної та мережевої моделі даних. URL: https://jait.donnu.edu.ua/article/view/12268 (дата звернення 27.09.2023).
5. What is Client-Server Architecture? Everything You Should Know. URL: https://www.simplilearn.com/what-is-client-server-architecture-article (дата звернення 12.09.2023).
6. Andrew Lock. ASP.NET Core in Action. New York : Manning Publications, 2023. 984 с.
7. Adam Freeman. Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. New York : Apress, 2022. 1286 с.
8. Adam Freeman. Pro ASP.NET Core Identity: Under the Hood with Authentication and Authorization in ASP.NET Core 5 and 6 Applications. New York : Apress, 2021. 746 с.

9. Mark J. Price. C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development: Build applications with C#, .NET Core, Entity Framework Core, ASP.NET Core, and ML.NET using Visual Studio Code. Birmingham : Packt, 2019. 820 с.
10. What is Entity Framework?. URL: https://www.entityframeworktutorial.net/entityframework6/what-is-entityframework.aspx (дата звернення 13.09.2023).
11. Jon P. Smith. Entity Framework Core in Action. New York : Manning Publications, 2021. 624 с.
12. MySQL Documentation. URL: https://dev.mysql.com/doc/ (дата звернення 13.09.2023).
13. Солодун Т. Р., Зелінська О. В. Помилки в системах баз даних і теорема CAP. URL: https://jait.donnu.edu.ua/article/view/12273 (дата звернення 27.09.2023).
14. Silvia Botros, Jeremy Tinley. High Performance MySQL: Proven Strategies for Operating at Scale. Sebastopol : O'Reilly Media, 2021. 386 с.
15. Розробка сайтів та програми на Angular. URL: https://wezom.com.ua/ua/blog/razrabotka-sajtov-i-prilozhenie-angular (дата звернення 14.09.2023).
16. Розбираємо таємницю Angular Dependency Injection. URL: https://dou.ua/forums/topic/44768/ (дата звернення 15.09.2023).
17. Angular проти React. URL: https://foxminded.ua/angular-vs-react/ (дата звернення: 16.09.2023).
18. Documentation for Visual Studio Code. URL: https://code.visualstudio.com/docs (дата звернення 18.09.2023).
19. Семенюк А. М., Зелінська О.В.. Обробка інформації з використанням реляційних баз даних. URL: https://jktod.donnu.edu.ua/article/view/13107 (дата звернення 27.09.2023).

20. Мисько Б.В., Петришин В.С., Зелінська О.В. Логічне проектування баз даних. URL: https://jktod.donnu.edu.ua/article/view/13101 (дата звернення 27.09.2023).
21. Демиденко М. А. Введення в сучасні бази даних: навч. посіб. Дніпро: НТУ «Дніпровська політехніка». 2020. 38 с.
22. Data-Access Layer. URL: https://code.visualstudio.com/docs (дата звернення 18.09.2023).
23. What I Love About Data Access Layer (DAL) When Building Small Backend Projects. URL: https://levelup.gitconnected.com/what-i-love-about-data-access-layer-dal-when-building-small-backend-projects-8ad694baee3a (дата звернення 19.09.2023).
24. What is the Business-Logic Layer?. URL: https://www.tutorialspoint.com/what-is-the-business-logic-layer (дата звернення 23.09.2023).
25. Creating a Business Logic Layer (C#). URL: https://learn.microsoft.com/en-us/aspnet/web-forms/overview/data-access/introduction/creating-a-business-logic-layer-cs (дата звернення 23.09.2023).
26. Розділяй та володарюй: що таке патерни MVC і MVP, та як їх використовувати. URL: https://highload.today/uk/blogs/shho-take-mvc-ta-mvp-patterni/ (дата звернення 25.09.2023).
27. Башовий В.М., Стаценко В.В., Стаценко Д.В. (2023). Визначення швидкості роботи сучасних фреймворків для створення web-інтерфейсів. Technologies and Engineering, (4), 9-16. doi: 10.30857/2786-5371.2022.4.1.
28. Oriyano Sean-Philip Penetration testing essentials. Indianapolis: John Wiley & Sons, Inc. 2017. 349 p.
29. Bertoglio D. D., Zorzo A. F. Overview and open issues on penetration test. Journal of the Brazilian Computer Society. 2017. №23. P. 1–16.

30. Порошин С. М., Можаєв О. О., Можаєв М. О. Методологія проведення pen-тестування вебдодатків. Системи обробки інформації. 2016. Випуск 3 (140). С. 33–35
31. Барибін О. І. Сучасні методології тестування на проникнення. Кібербезпека у системі національної безпеки України: пріоритетні напрями розвитку: збірник матеріалів наукового круглого столу, м. Маріуполь, 26 квітня 2018 р. С. 63–66.
32. ДСТУ ISO/IEC 25010:2016 (ISO/IEC 25010:2011, IDT) Інженерія систем і програмних засобів. Вимоги до якості систем і програмних засобів та її оцінювання (SQuaRE). Моделі якості системи та програмних засобів.
33. Halton W., Weaver B., Ansari J. A. Penetration Testing: A Survival Guide. Birmingham: Packt Publishing Ltd. 2016. 1045 p
34. Mirjalili M., Nowroozi A., Alidoosti M. A survey on web penetration test. Advances in Computer Science: an International Journal. 2014. № 3. P. 107–121.
35. Барибін О.І. Методологія тестування на проникнення веб-сайту закладу вищої освіти. URL: https://cutt.ly/TRDGgFz (дата звернення 28.09.2023).
36. Đorđević N. Evaluation of the usability of Web-based applications. Vojnotehnički glasnik / Military technical courier. Vol. 65. Issue 3, 2017, pp.785–802.
37. Gledec G. Quality Model for the World Wide Web. 8th International Conference on Telecommunications – ConTEL2005. Zagreb, Croatia, June 2005, pp 281–287.

Додаток 2 до наказу
від «31» березня 2023 року
№119/05

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____
_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____

_____

(дата) (підпис)