m-math_2025_005

Міністерство освіти і науки України

Чернівецький національний університет

імені Юрія Федьковича

Факультет математики та інформатики
(повна назва інституту/факультету)

Кафедра математичного моделювання
(повна назва кафедри)

Розробка додатка організації туристичного сервісу з використанням інтелектуального аналізу даних та з інтегрованим телеграм-ботом

Кваліфікаційна робота

Рівень вищої освіти - другий (магістерський)

Виконав:
студент 6 курсу, групи 601
спеціальності 122 – Комп'ютерні науки_
(назва спеціальності)
Клейман Максим Олександрович_____
(прізвище, ім'я та по-батькові)
Керівник доцент Пасічник Г.С._____
(науковий ступінь, вчене звання, прізвище та ініціали)

До захисту допущено:

Протокол засідання кафедри № __

від „___" грудня 2024 р.

зав. кафедри _____ проф. Черевко І.М.

Чернівці – 2024

АНОТАЦІЯ

Розроблено вебсайт туристичного агентства з функціями агрегатора турів і рекомендаційної системи. Система дозволяє переглядати турпакети, бронювати тури, отримувати рекомендації на основі вподобань і виконувати CRUD-операції для турів (адмінпанель). Авторизацію реалізовано засобами JWT Token, рекомендації – засобами SVD, Pandas, NumPy, SQLAlchemy та TfidfVectorizer. Реалізовано Telegram-бот для автоматизації створення й надсилання актів звірки, що підвищує зручність користувачів.

Ключові слова: рекомендаційна система, телеграм-бот, вебсайт, електронний сервіс, мови програмування Java, Python, BAF, реляційна база даних, Spring Boot, Angular, Flask, Angular Material, алгоритм k-NN.

ABSTRACT

A website for a travel agency has been developed, featuring a tour aggregator and recommendation system. The platform allows users to view tour packages, book tours, receive personalized recommendations, and perform CRUD operations for tours (admin panel). Authorization is implemented via JWT Token, and the recommendation system is built using SVD, Pandas, NumPy, SQLAlchemy, and TfidfVectorizer. Additionally, a Telegram bot has been integrated to automate the creation and sending of reconciliation statements, enhancing user convenience.

Keywords: recommendation system, Telegram bot, website, electronic service, programming languages Java, Python, BAF, relational database, Spring Boot, Angular, Flask, Angular Material, k-NN algorithm.

Кваліфікаційна робота містить результати власних досліджень, а також посилання на джерела, де використано ідеї чи результати інших авторів.

_____ М.О. Клейман

ЗМІСТ

ВСТУП

Людина завжди прагне до пізнання нового: відкривати інновації, засвоювати нову інформацію, знайомитися з культурою та традиціями інших народів. Одним із найбільш захопливих способів реалізації цієї потреби є туризм. Попри сучасну епоху інформаційних технологій, яка надає доступ до знань без виходу з дому, подорожі залишаються унікальним досвідом. Тільки мандрівник, який відвідує нові місця, пізнає культуру інших народів і змінює своє світосприйняття, може по-справжньому відчути багатство світу. Як зазначено у літературі [1], туризм — це не лише переміщення у просторі, а й трансформація особистості, яка відкривається новому.

Сьогодні штучний інтелект (ШІ) активно впроваджується у різні сфери життя. Він став важливою складовою сучасних інженерних рішень, що здатні обробляти дані, аналізувати їх і застосовувати отримані знання. ШІ використовується у будівництві, ігровій індустрії, освіті та багатьох інших галузях. Одним із найбільш помітних застосувань ШІ є створення чат-ботів — програм, які здатні імітувати діалог з людиною.

Туризм — це сфера, де ШІ може стати ключовим фактором у підвищенні якості послуг. Зростаюча кількість людей шукає ефективні способи організації подорожей, що спонукає до розвитку сучасних туристичних систем. Для задоволення таких потреб було розроблено туристичний агрегатор у вигляді вебсервісу, який інтегрує функціонал рекомендаційної системи.

Актуальність тематики роботи полягає в необхідності створення сучасних зручних інструментів для планування подорожей. Багато людей стикаються з труднощами під час пошуку та бронювання туристичних послуг через відсутність персоналізованого підходу або недостатню інформацію.

Традиційні туристичні агенції часто не відповідають очікуванням сучасних клієнтів, а існуючі онлайн-ресурси можуть бути ненадійними. Розробка інтегрованої системи допоможе вирішити ці проблеми та задовольнити потреби користувачів. Мета роботи полягає у створенні

вебсайту туристичного агентства з розширеними функціями туристичного агрегатора та рекомендаційної системи.

Створений проєкт дозволяє:

- надати користувачам можливість перегляду туристичних пакетів з фільтрацією за обраними параметрами;
- забезпечити авторизацію та перегляд персонального профілю;
- реалізувати функцію замовлення турів;
- надати працівникам агентства можливість редагувати, видаляти та додавати туристичні пакети;
- відображати інформацію про тур, включно з атракціями та місцями відпочинку;
- запровадити функцію персоналізованих рекомендацій турів на основі вподобань користувача;
- інтегрувати Telegram-бот для автоматизації управління документами, включаючи створення та надсилання актів звірки.

Об’єктом дослідження є система туристичного агентства з розширеними функціями та інтеграцією рекомендаційної системи.

Предметом дослідження є методи спрощення взаємодії користувачів із системою, підвищення ефективності вибору та бронювання туристичних послуг через впровадження інноваційних технологій, зокрема Telegram-ботів.

Робота складається з 4 розділів. Теоретична частина містить огляд сучасних технологій розробки електронних сервісів, аналіз аналогів з ринку онлайн-послуг, опис засобів створення вебдодатків з широкими можливостями. Практична частина містить опис власноруч створеного проєкту, його можливості та технології розробки.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ТЕХНОЛОГІЙ РОЗРОБКИ

1.1. Аналіз предметної області

Туризм сьогодні є однією з найважливіших і найдинамічніших галузей економіки. За останні десятиліття ця сфера набула значної популярності серед жителів різних країн, що спричинило стрімке зростання кількості міжнародних подорожей і розширення ринку туристичних послуг [1].

Одним із ключових факторів, які стимулюють розвиток туризму, є швидкий технологічний прогрес і широке поширення інтернету. Доступ до Інтернету дозволяє туристам миттєво отримувати інформацію про цікаві місця, готелі та туристичні об’єкти. Завдяки цьому процес планування подорожей стає набагато зручнішим, а онлайн-бронювання готелів і квитків стає звичним явищем.

Розвиток соціальних мереж та поширення мобільних пристроїв також сприяють популяризації подорожей. Люди охоче діляться своїми враженнями, фото та рекомендаціями в соцмережах, що мотивує інших до планування власних подорожей.

Розробка сучасної системи, яка поєднує функціонал туристичного агрегатора та рекомендаційної системи, є актуальним завданням для задоволення потреб сучасних туристів. Такий підхід дозволяє інтегрувати дані з різних джерел (авіакомпанії, готелі, туристичні компанії) та забезпечити клієнтів інструментами для порівняння цін і умов бронювання.

Рекомендаційна система забезпечує персоналізований підхід до підбору туристичних пакетів. Використовуючи алгоритми аналізу даних і машинного навчання, система надає індивідуальні пропозиції, що враховують переваги користувачів, їхній бюджет і попередній досвід. Це підвищує зручність користування сервісом і сприяє створенню позитивного користувацького досвіду.

У рамках проєкту також було створено Telegram-бот, який автоматизує створення та надсилання актів звірки для клієнтів. Бот розроблений на мові BAF і інтегрований із внутрішньою базою даних системи. Його функціонал дозволяє:

- автоматично створювати акти звірки за заданими параметрами;
- відправляти їх клієнтам через Telegram у форматі PDF;
- оновлювати дані про звірки у базі системи.

Цей бот значно полегшує обробку документів, мінімізуючи ручну роботу та скорочуючи час обробки запитів. Для створення програмного забезпечення використано такі інструменти:

- Spring Boot для бекенд-розробки, що дозволило створити масштабовану та стабільну серверну частину;
- Angular для фронтенду, забезпечуючи інтерактивність та зручність інтерфейсу;
- Python для реалізації рекомендаційної системи з використанням бібліотек Pandas, NumPy, і scikit-learn;
- Flask для обміну даними між рекомендаційною системою та основним додатком;
- BAF для створення Telegram-бота.

Переваги створеної системи полягають в інтеграції функціоналу агрегатора, рекомендаційної системи та Telegram-бота дозволяє забезпечити:

- персоналізований підбір турів для клієнтів;
- швидкий доступ до інформації про туристичні послуги;
- зручне управління документами.

Саме тому розробка такого туристичного агентства відповідає сучасним вимогам суспільства. Поєднання рекомендаційної системи, автоматизації роботи з документами та зручного інтерфейсу дозволяє створити ефективний інструмент для планування подорожей, який задовольняє потреби користувачів і забезпечує комфортний туристичний досвід.

1.2. Аналіз аналогів

Для проведення аналізу предметної області необхідно дослідити існуючі аналоги програмних рішень. Це дозволяє виявити їхні сильні та слабкі сторони, що є основою для покращення функціональності та якості власного програмного забезпечення.

Одним із найпоширеніших аналогів є вебсайт TripAdvisor, а також чат-бот Travel Helper, доступний у месенджері Telegram.

TripAdvisor вже давно займає лідерські позиції серед ресурсів для планування подорожей. Його основною перевагою є великий обсяг корисної інформації для мандрівників. Користувачі можуть знайти на цьому сайті готелі, заклади харчування, розваги, а також популярні місця для подорожей, базуючись на вибраному напрямку. Окрім того, ресурс надає рекомендації щодо туристичних локацій і навіть формує рейтинги, що спрощує прийняття рішення.

Travel Helper у Telegram надає простий і швидкий спосіб отримання інформації через зручний інтерфейс месенджера. Основні функції бота включають пошук туристичних послуг, відображення популярних місць і рекомендації на основі введених користувачем даних. Однак його функціонал обмежений, особливо якщо порівнювати з комплексними можливостями вебсайтів.

Результати аналізу вказують на можливість об'єднання кращих функцій обох рішень у єдиній системі. Наприклад, можна інтегрувати можливості TripAdvisor із простотою використання чат-бота. Це дозволить створити зручний сервіс, який забезпечить як багатофункціональний інтерфейс, так і швидкість та доступність месенджера.

На рис. 1.1 продемонстровано інтерфейс вебсайту TripAdvisor, який є прикладом вдосконалення елементів користувацького інтерфейсу розробленої системи.

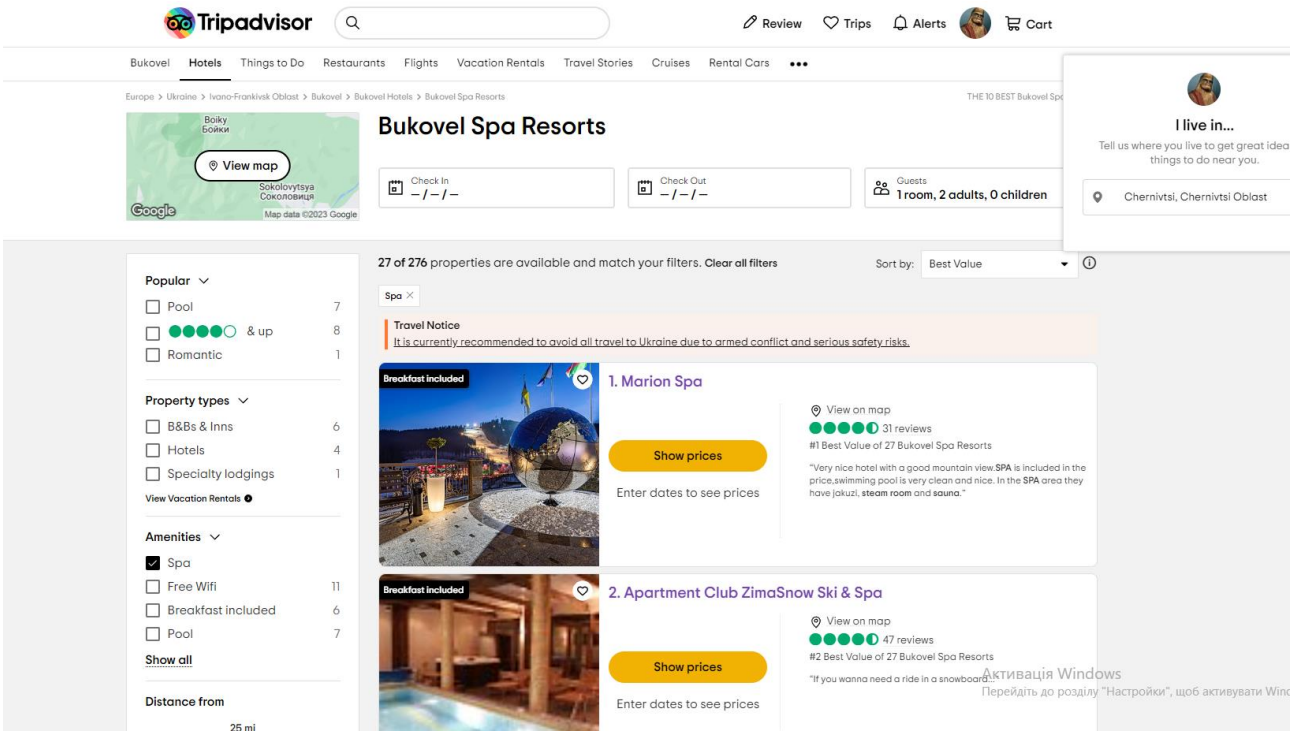


Рис. 1.1 Вебсайт Trip Advisor

Переваги розглядуваного сайту полягають в:

- зручному пошуку за геолокацією, що дозволяє швидко знайти місця відпочинку поблизу;
- інтуїтивно зрозумілому і привабливому інтерфейсу, який полегшує навігації;
- організації місць відпочинку за локаціями, що сприяє зручному перегляду варіантів.

Основними недоліками сайту TripAdvisor, на мою думку, є:

- відсутність можливості безпосереднього бронювання туристичних турів через платформу;
- брак функціональності для аналізу даних і формування персоналізованих рекомендацій.

Telegram-бот «TravelHelper» створений для користувачів, які планують дослідити цікаві місця поблизу. Користувач надсилає своє місцезнаходження та вказує бажану відстань для пошуку. На основі цієї інформації бот пропонує список різноманітних закладів, пам’яток, спортивних об’єктів, автоматично розраховуючи їхню кількість.

При виборі одного з пунктів користувач може або перейти до додаткових підкатегорій, або одразу отримати доступ до меню вибраних туристичних місць. У цьому меню є можливість детально ознайомитися з інформацією про об'єкти або переглянути їхню точну локацію.

Інтерфейс Telegram боту Travel Helper зображений на рис. 1.2

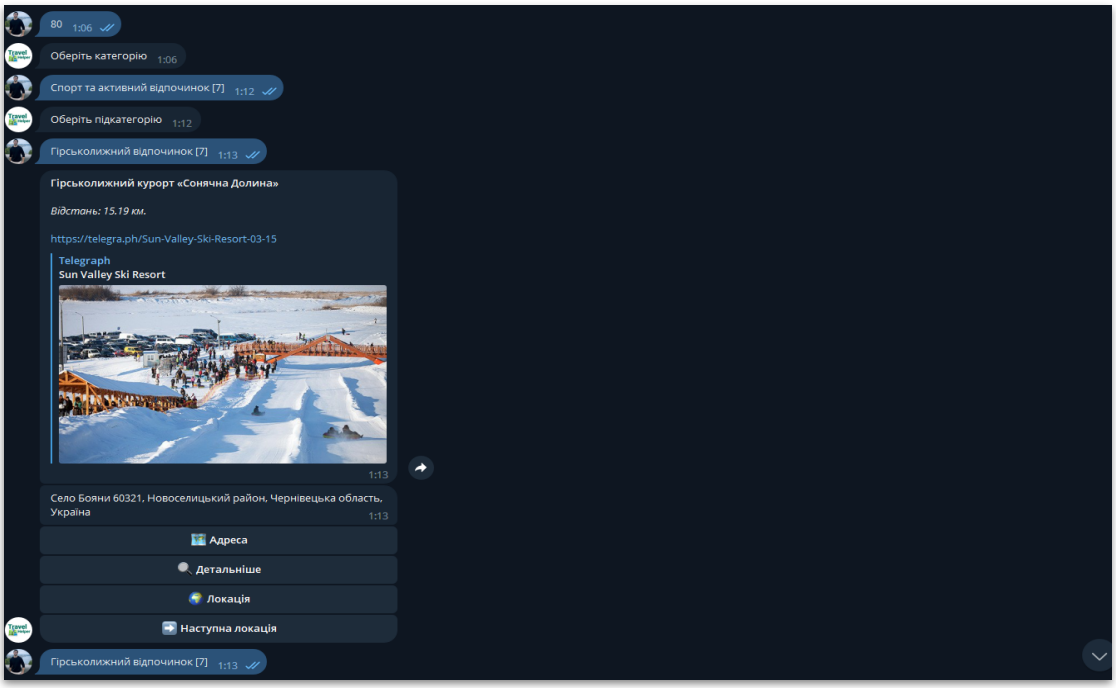


Рис. 1.2 Telegram Bot Travel Helper

Сильними сторонами чат-бота Travel Helper є:

– зручний доступ через платформу Telegram;

– висока швидкість обробки запитів;

– інтуїтивно зрозумілий та зручний для користувача інтерфейс;

– можливість здійснювати пошук цікавих місць для подорожей на основі геолокації.

Слабкими сторонами чат-бота Travel Helper є:

– обмежений набір функцій;

– досить простий візуальний дизайн інтерфейсу;

– відсутність механізмів інтелектуального аналізу даних;

– неможливість організувати бронювання чи замовлення туру через бот.

Аналіз аналогічних програмних рішень показав, що наявні сервіси частково відповідають поставленим вимогам, проте не здатні повністю реалізувати всі завдання, які передбачені цією кваліфікаційною роботою.

РОЗДІЛ 2. АНАЛІЗ СТВОРЕНОГО ДОДАТКА

2.1. Опис та специфікація вимог

Кожен вебдодаток поєднує серверну частину, яка виконує обробку запитів, і клієнтську частину, що взаємодіє з користувачем для отримання та обробки даних у зручному вигляді. На етапі створення вебдодатку особливу увагу потрібно приділити визначенню функціональних вимог і можливостей, які будуть реалізовані в рамках проєкту.

Одним із важливих етапів розробки програмного забезпечення є укладення угоди між замовником і розробником, де детально прописуються завдання і функціонал майбутнього продукту. Це дозволяє забезпечити створення якісного, повноцінного рішення, що відповідатиме очікуванням клієнтів.

У межах кваліфікаційної роботи передбачено створення інтелектуального сервісу для туристичного агентства з можливостями туристичного агрегатора.

Структура розробленого мною проєкту включає:

– базу даних у MySQL;

– серверну частину на Spring Boot;

– клієнтську частину, реалізовану за допомогою Angular;

– рекомендаційну систему, створену мовою Python.

Також передбачено інтеграцію Telegram-бота, що забезпечує автоматизацію додаткових функцій, зокрема створення актів звірки. Telegram-бот написаний з використанням мови BAF і призначений для взаємодії з бухгалтерськими системами.

Основні можливості створеного бота:

– формування актів звірки на основі запитів користувачів;
– автоматичне надсилання сформованих документів у форматі PDF;
– можливість обробки запитів на генерацію актів звірки для конкретних контрагентів;
– зручний інтерфейс для отримання готових документів у месенджері.

Основні вимоги до проекту такі:

– перегляд і пошук туристичних пакетів;
– функції авторизації та реєстрації користувачів;
– замовлення турів через вебдодаток;
– перегляд історії замовлень для авторизованих користувачів;
– можливість виконання CRUD-операцій із турами через адміністративну панель;
– інтеграція туристичного агрегатора для зручності вибору пропозицій;
– рекомендаційна система для персоналізації пошуку турів;
– використання Telegram-бота для формування та надсилання актів звірки.

2.2. Функціонал додатка

Розроблена система забезпечує наступні можливості для різних категорій користувачів.

1. Зареєстровані користувачі:
   - перегляд туристичних пропозицій;
   - редагування власного профілю;
   - перегляд історії своїх замовлень;
   - замовлення турів;
   - отримання персоналізованих рекомендацій на основі вподобань.
2. Незареєстровані користувачі:
   - можливість переглядати туристичні пропозиції.

3. Адміністратори:
    - усі функції зареєстрованого користувача (за винятком редагування профілю);
    - перегляд усіх замовлених турів;
    - виконання CRUD-операцій із турами.

Вхідні дані для роботи з додатком:

- база даних туристичного агентства;
- обрані користувачем тури (для роботи рекомендаційної системи).

Вихідними даними є:

- відображення туристичних пакетів;
- можливість замовлення туру;
- реалізація авторизації користувачів;
- надання рекомендацій турів;
- виведення історії замовлень користувача;
- детальна інформація про місце проведення туру (можливості туристичного агрегатора).

Особливістю системи є інтеграція Telegram-бота, що виконує важливі додаткові функції, зокрема автоматизацію документообігу. Бот створений на базі мови BAF і забезпечує:

- формування актів звірки для користувачів, які працюють із фінансовими чи комерційними запитами;
- генерацію документів у форматі PDF на основі даних із бази;
- надсилання готових актів звірки безпосередньо у Telegram користувачу;
- інтерактивний інтерфейс, що дозволяє швидко та зручно запитувати та отримувати документи.

Основні типи задач системи, що виконує програмне забезпечення:

- реєстрація користувачів;
- рошук турів;
- перегляд деталей турів;

- бронювання турів;
- керування акаунтом користувача;
- взаємодія з інформацією про тури з боку адміністратора;
- перегляд історії замовлень;
- надання рекомендацій турів.

Принциповими задачами є організація бронювання та реалізація турів; аналіз потреб клієнтів; пошук і укладення контрактів із постачальниками послуг.

Сервісними задачами є забезпечення зручної навігації між режимами роботи додатку; створення інтуїтивно-зрозумілого користувацького інтерфейсу; розробка бази даних для взаємодії із серверною частиною додатку та її інтеграція з клієнтським інтерфейсом.

Вебсайт розроблений із використанням Spring Boot для серверної частини та Angular для клієнтської. Telegram-бот доповнює функціональність системи, створюючи інструмент для інтерактивної взаємодії та автоматизації задач.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

3.1. Опис алгоритму

Метод SVD (сингулярного розкладу матриць) є популярним інструментом для роботи з матрицями, зокрема для виявлення їхніх сингулярних значень та векторів. Цей метод знаходить застосування в численних сферах, таких як машинне навчання, аналіз даних та системи рекомендацій.

У рекомендаційних системах SVD використовується для розкладу матриці оцінок, яка відображає взаємодію користувачів із предметами, на три компоненти. Такий підхід дає змогу визначити приховані патерни у взаємодіях, спрощуючи аналіз і покращуючи якість прогнозів.

Сутність застосування методу SVD у рекомендаційних системах полягає у використанні лише кількох найбільших сингулярних значень для апроксимації початкової матриці. Це сприяє зменшенню розмірності та виокремленню ключових факторів, які впливають на формування рекомендацій.

Кроки алгоритму SVD для рекомендаційних систем можуть бути наступними.

1. Побудова матриці оцінок: Спочатку формується матриця оцінок, де кожен елемент матриці представляє оцінку користувача для певного предмета. Якщо користувач не зробив оцінку, елемент може бути позначений відсутнім значенням або нулем.

2. Розклад матриці: Застосовується алгоритм SVD для розкладу матриці оцінок на добуток трьох матриць: U, Σ і V^T. Матриця U представляє користувачів і містить інформацію про їхній профіль, матриця Σ є діагональною матрицею сингулярних значень, які відображають важливість факторів, а матриця V^T представляє предмети і містить інформацію про їхні властивості.

3. Відбір головних факторів: Обираються перші N найбільших сингулярних значень та відповідні стовпці в матрицях U і V^T, де N - це задана кількість головних факторів. Це дозволяє зменшити розмірність матриць і виокремити найбільш значущі аспекти взаємодії між користувачами і предметами.

4. Генерація рекомендацій. Після отримання скороченого розкладу матриці,можна використовувати його для генерації рекомендацій. Це може бути здійснено, наприклад, шляхом обчислення подібності між користувачами або предметами на основі їхніх профілів і використання цієї інформації для рекомендацій нових предметів користувачам.

Метод SVD дозволяє створювати персоналізовані рекомендації, враховуючи взаємозв’язки між користувачами та предметами. Його активно застосовують у різних сферах: від електронної комерції та соціальних мереж

до стримінгових платформ, щоб підвищити точність рекомендацій і покращити користувацький досвід.

3.2. Обґрунтування вибору алгоритму

Метод SVD є одним із ефективних інструментів для створення рекомендаційних систем завдяки простоті реалізації та здатності враховувати складні взаємозв'язки між користувачами і предметами.

Переваги методу SVD [2]:

1) персоналізація рекомендацій (метод дозволяє створювати індивідуальні рекомендації, виявляючи приховані залежності між користувачами і предметами, які недоступні для традиційних підходів);

2) робота з розрідженими даними (SVD здатний заповнювати пропуски в даних і пропонувати рекомендації навіть для нових користувачів або предметів, для яких мало інформації);

3) оптимізація обчислень: Зменшення розмірності матриць U, Σ і V^T дозволяє знизити обчислювальну складність та вимоги до ресурсів, що робить SVD ефективним для обробки великих обсягів даних.

Недоліки методу SVD такі:

1) високі обчислювальні витрати (для великих матриць розрахунок SVD може бути ресурсомістким і потребувати значних обсягів пам'яті, що обмежує його використання в реальному часі);

2) ризик перенавчання (недостатня кількість даних або надто складна модель можуть спричинити перенавчання, що знижує точність рекомендацій для нових даних);

3) відсутність урахування контексту (етод не враховує додаткові фактори, такі як час, місце чи індивідуальні вподобання користувачів, що може обмежувати якість рекомендацій у специфічних умовах).

Розглянемо інші методи побудови рекомендаційних систем.

1. TF-IDF (Term Frequency-Inverse Document Frequency). Його переваги: простота реалізації, добре працює з текстовими даними, враховуючи важливість слів. Його недоліки: ігнорує контекст та порядок даних, не підходить для сильно розріджених даних.

2. Cosine Similarity. Його перваги: легкість у використанні, враховує схожість між об’єктами за кутом між їх векторними представленнями, підтримує числові, текстові та категоріальні дані. Його недоліки: не враховує послідовність даних, схожість нульового рівня може ускладнювати інтерпретацію.

3. TF-DNN (Term Frequency-Deep Neural Networks). Його переваги: автоматично визначає значущі ознаки, підходить для складних моделей. Його недоліки: потребує великих обчислювальних ресурсів і значного часу на навчання, вимагає великої кількості даних для ефективного функціонування.

На рис. 3.1 представлено приклад роботи алгоритму SVD, що ілюструє основні етапи його застосування.

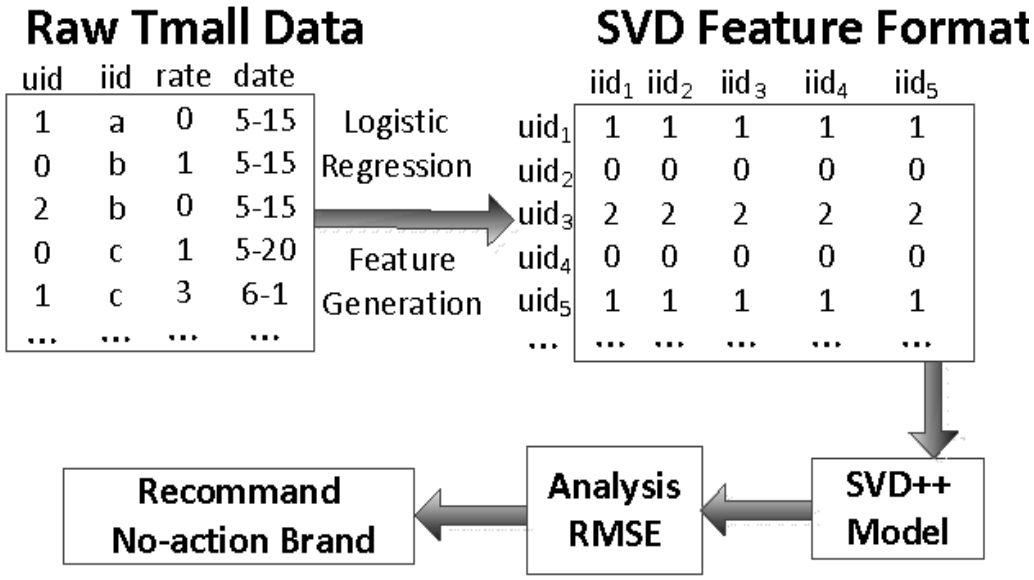


Рис. 3.1 Приклад роботи алгоритму SVD

З урахуванням описаних вище альтернатив було прийнято рішення розробити рекомендаційну систему, базуючись на алгоритмі SVD, через його простоту в реалізації та здатність забезпечувати високу персоналізацію рекомендацій [10].

3.3. Реалізація та результат роботи рекомендаційної системи

Як вже було зазначено, методом для створення рекомендаційної системи було використано SVD. Для реалізації методу спочатку потрібно було підготувати дані. Для цього ми, використовуючи бібліотеку SqlAlchemy, витягуємо дані таблиці Tour зі створеної до проекту бази даних, а також на основі дій користувачів складемо рейтинг турів.

Код, який відповідає за підготовку даних описано нижче:

```
db_connection_str  = 'mysql+pymysql://ymsteam:' + 'team123456' + '@localhost:3306/touragency'
db_connection = create_engine(db_connection_str)
df = pd.read_sql('SELECT * FROM tour', con = db_connection)
df.to_dict()
connect = db_connection.connect()
frame = pd.read_sql("select * from tour", connect);
pd.set_option('display.expand_frame_repr', False)
print(frame)
ratings_data = pd.read_csv('rating.csv')
ratings_data.head(10)
from surprise import Dataset
from surprise import Reader
import surprise
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(ratings_data[['place_id', 'user_id', 'rating']], reader)
```

Після попередньої підготовки даних (згідно з раніше наведеним кодом) потрібно встановити метрику та алгоритм для навчання моделі. Після цього можна перейти безпосередньо до навчання моделі.

Код, який відповідає за встановлення алгоритму та навчання моделі описано нижче:

```
from surprise import SVD
from surprise.model_selection import cross_validate
svd = SVD(verbose=True, n_epochs=10)
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=3, verbose=True)
trainset = data.build_full_trainset()
svd.fit(trainset)
```

Для отримання власне самої рекомендації, використаємо наступний код та згенеруємо рекомендований тур:

```
import random
import difflib
def get_tour_id(tour_title, metadata):
    existing_titles = list(metadata['name'].values)
    closest_titles = difflib.get_close_matches(tour_title, existing_titles)
    book_id = metadata[metadata['name'] == closest_titles[0]]['id'].values[0]
    return book_id
def get_tour_info(tour_id, metadata):
    tour_info = metadata[metadata['id'] == tour_id][['id', 'name',
                                        'description', 'price']]
    return tour_info.to_dict(orient='records')
def predict_review(user_id, tour_title, model, metadata):
    tour_id = get_tour_id(tour_title, metadata)
    review_prediction = model.predict(uid=user_id, iid=tour_id)
    return review_prediction.est

def generate_recommendation(user_id, model, metadata, thresh=4):
    tour_titles = list(metadata['name'].values)
    random.shuffle(tour_titles)
    for tour_title in tour_titles:
        rating = predict_review(user_id, tour_title, model, metadata)
        if rating >= thresh:
            tour_id = get_tour_id(tour_title, metadata)
            return get_tour_info(tour_id, metadata)
```

На рис. 3.2 зображений вивід рекомендації для користувача під номером 2:

```
In [73]: generate_recommendation(2, svd, frame)

Out[73]: [{'id': 3,
           'name': 'Nevo by Isrotel Collection 5*',
           'description': 'Готель розташований через дорогу від власного піщаного пляжу біля Мертвого моря. Побудований у 1985 році, ост
         анню реставрацію було проведено у 2010 році.',
           'price': 150000.0}]

         Візуалізація
```

Рис. 3.2 Виведення рекомендації для певного користувача

РОЗДІЛ 4. РОБОЧИЙ ПРОЄКТ

4.1 Засоби розробки

Розглянемо засоби розробки, які були обрані для даного програмного продукту, зокрема для інтеграції Telegram-бота, який відправляє акти звірки у форматі PDF за запитом користувача.

Spring Boot — це фреймворк для розробки Java-додатків, побудований на основі платформи Spring. Завдяки автоматичній конфігурації та конвенційним налаштуванням, Spring Boot спрощує розробку, конфігурацію та розгортання додатків. Фреймворк підтримує вбудовані сервери, як-от Tomcat, Jetty або Undertow, що дозволяє запускати додатки без окремого налаштування сервера. Spring Boot також інтегрується з системами управління залежностями, такими як Maven або Gradle, і надає можливість роботи з різними базами даних.

Angular — потужний фреймворк для розробки вебдодатків, який використовує TypeScript, що додає сильну типізацію і об’єктно-орієнтовані можливості. Angular дозволяє розбивати додаток на незалежні компоненти, що полегшує розробку та тестування. Окрім того, фреймворк підтримує маршрутизацію, валідацію форм, аутентифікацію та обробку HTTP-запитів, що робить його ідеальним вибором для створення складних односторінкових додатків.

Python — універсальна мова програмування, яка широко використовується в таких сферах, як веброзробка, штучний інтелект і аналіз даних. Її популярність зумовлена простотою синтаксису та наявністю великої кількості бібліотек для різноманітних завдань. Для розробки

Telegram-ботів використовують Python завдяки зручним бібліотекам, таким як python-telegram-bot, яка дозволяє швидко інтегрувати бота в різні системи. Для генерації PDF-документів, зокрема актів звірки, використовуються бібліотеки, як-от FPDF або ReportLab, що дають можливість створювати PDF-файли на основі даних і відправляти їх користувачам через Telegram-бота.

Flask — легкий Python-фреймворк для створення вебдодатків. Завдяки своїй простоті та гнучкості, Flask є ідеальним для створення серверної частини, що взаємодіє з Telegram-ботом і надає функціональність для обробки запитів від користувачів. Цей фреймворк підтримує роботу з маршрутами, шаблонізацією і HTTP-запитами, що дозволяє швидко налаштувати сервер для взаємодії з Telegram.

MySQL — популярна реляційна система управління базами даних, яка використовується для зберігання та обробки великих обсягів структурованої інформації. Вона забезпечує високу продуктивність і надійне збереження даних. MySQL є ідеальним рішенням для зберігання інформації про користувачів та їхні запити на акти звірки, що дозволяє Telegram-боту ефективно працювати з великими обсягами даних.

Telegram-бот — цей бот, реалізований на мові **BAF**, є частиною системи і має функціональність для приймання запитів від користувачів щодо актів звірки. Бот взаємодіє з користувачами через Telegram, приймаючи їх запити і надсилаючи акти звірки у форматі PDF. Користувач може отримати необхідний акт звірки, запитавши його через бот, і отримати файл безпосередньо на свій пристрій. Для цього бот взаємодіє з сервером, отримує дані, генерує PDF-документ за допомогою спеціалізованих бібліотек і відправляє його через Telegram.

Інтеграція Telegram-бота з цією системою дозволяє створити інтерфейс, через який користувачі можуть запитувати акти звірки. Бот надає зручний канал для отримання документів за запитом без необхідності додаткових дій з боку користувача.

4.2 Інструкція програмісту

Мінімальні вимоги для програміста: знання СУБД MySQL та мови запитів SQL, знання мови Java та фреймворку Spring Boot, розуміння Angular та REST API, знання мови Python та базове розуміння фреймворку Flask.

Для початку потрібно встановити MySQL Server, запустити його, створити нового користувача, надайти йому привілегій та після входу у профіль користувача створити нову базу даних tourAgency.

Код з реалізованим вище діями описано нижче:

```
CREATE USER 'ymsteam'@'localhost' IDENTIFIED BY 'team123456';

GRANT ALL PRIVILEGES ON *.* TO 'ymsteam'@'localhost';

FLUSH PRIVILEGES;

mysql -uymsteam -pteam123456

CREATE DATABASE IF NOT EXISTS tourAgency;
```

Наступним кроком буде запуск серверної частини на Spring Boot та користувацької частини на Angular, за допомогою команди **ng serve**.

Для запуску частини проекту, яка стосується рекомендаційної системи, потрібно запустити сервер на Flask та встановити відповідні бібліотеки на мові Python.

Після завершення всіх цих етапів можна переходити, безпосередньо, до розробки.

Структура файлів проекту зображена на рис. 4.1

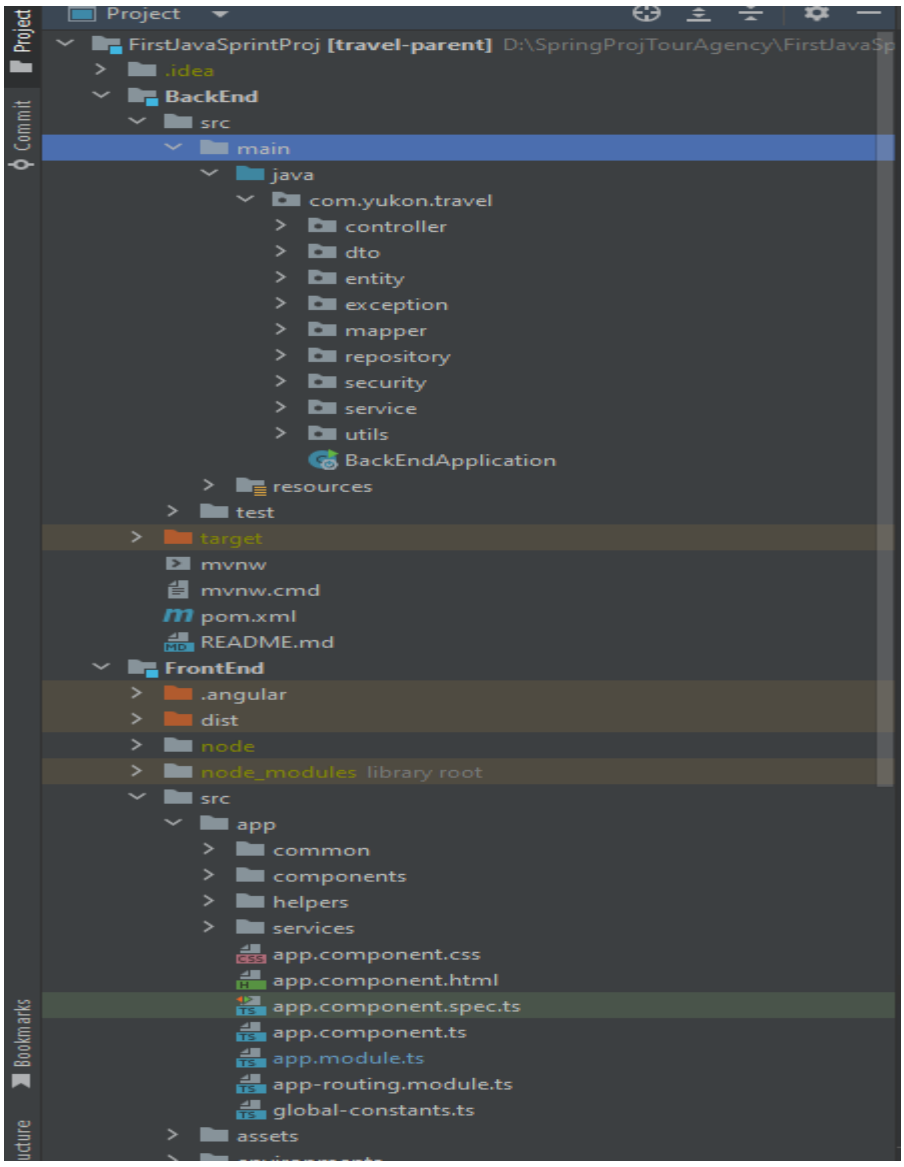


Рис. 4.1 Структура проєкту

Вміст вищевказаного проекту складається з Back-End та Front-End частини, перша виконана з використанням Spring Boot, друга – з використанням Angular.

Вміст Back-End частини складається з таких складових:

– controller – контролери для всіх сутностей;
– dto – dto для всіх сутностей;
– entity – сутності, які представляють таблиці у БД;
– exception – файл з виключеннями;
– mapper – mapper для всіх сутностей, які виконують request та response;
– repository - репозиторії для кожної сутності;

– security – виконання реєстрації та авторизації на серверній стороні;
– service – сервіси та їх імплементації для кожної з сутностей;
– utils – інші файли.

Вміст Front-End частини складається з таких складових:

– Common – класи для всіх сутностей на серверній частині та додаткові класи для певних реалізацій;
– Components – компоненти;
– Helpers – засоби для авторизації на користувацькій частині та валідатори;
– Services – сервіси для реалізації запитів до серверної частини.

Модель роботи програмного забезпечення виглядає так:

1) запускаємо MySQL Server використовуючи команду:

mysld –console

2) запускаємо Spring Boot, використовуючи файл BackEndApplication.java.

3) запускаємо Angular командою;

ng serve

4) переходячи, наприклад, на сторінку турів, клас tour.service робить Get запит до сервера Spring Boot;

5) сервер приймає запит у контролер;

6) контролер звертається до сервісів на Back End частині, в даному випадку, до сервісу TourService, який робить певні операції з даними, які прийшли з Front End частини, наприклад створює новий об’єкт, оновлює, фільтрує, сортує або, як у нашому випадку, повертає всі рядки таблиці;контролер, при успішному виконанні, виходить з методу сервісу таповертає відповідь, яку програміст виводить у HTML шаблон.

На рис. 4.2 зображено приклад роботи вебсайту з виведенням на екран турів з бази даних.

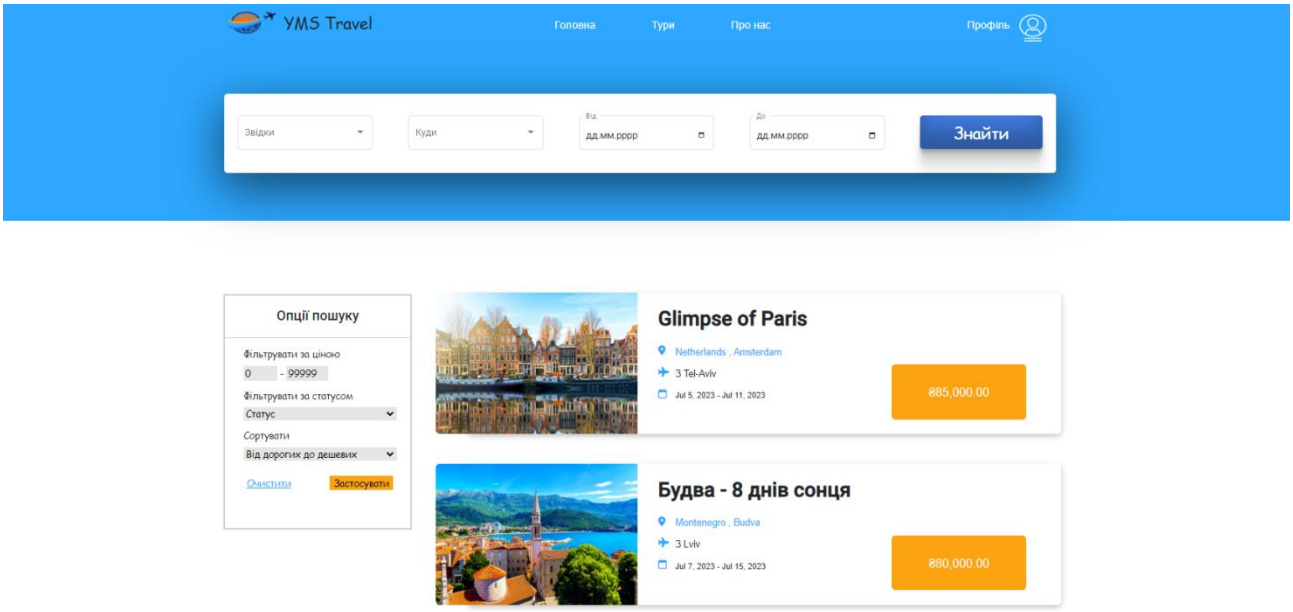


Рис. 4.2 Сторінка турів створеного проєкту

4.3 Інструкція користувачу

Для початку роботи над проектом потрібно запустити MySQL Server, Spring Boot та Angular. Інструкцію для цих дій було представлено у розділі 4.2.

Результатом виконання команд для старту програмного продукту зображений на рис. 4.3

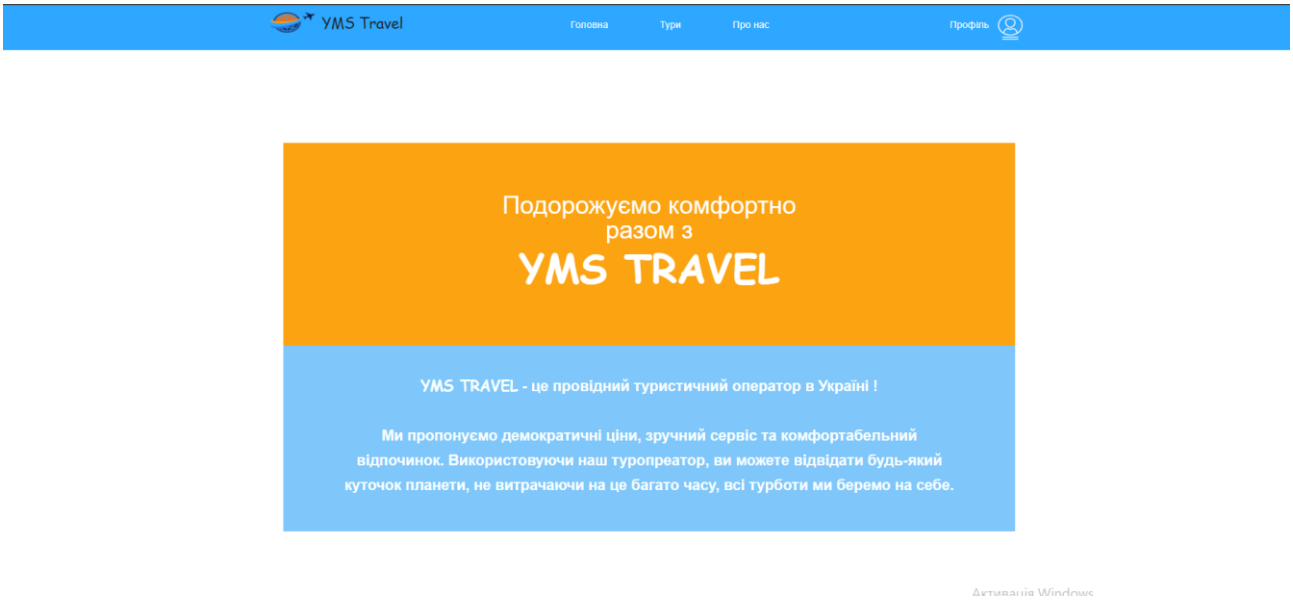


Рис. 4.3 Результат початкового запуску програмного продукту

З головної сторінки користувач отримує доступ до трьох розділів програмного застосунку «Тури», «Про нас» та «Профіль».

Перейшовши за посиланням «Про нас» користувачу надається інформацію про туристичне агенство, послуги, які він надає та його переваги. З сторінки «Про нас» ми можемо перейти на сторінку «Тури» клікнувши на клавішу «Почати пошук». Результат роботи сторінки «Про нас» зображений на рис. 4.4

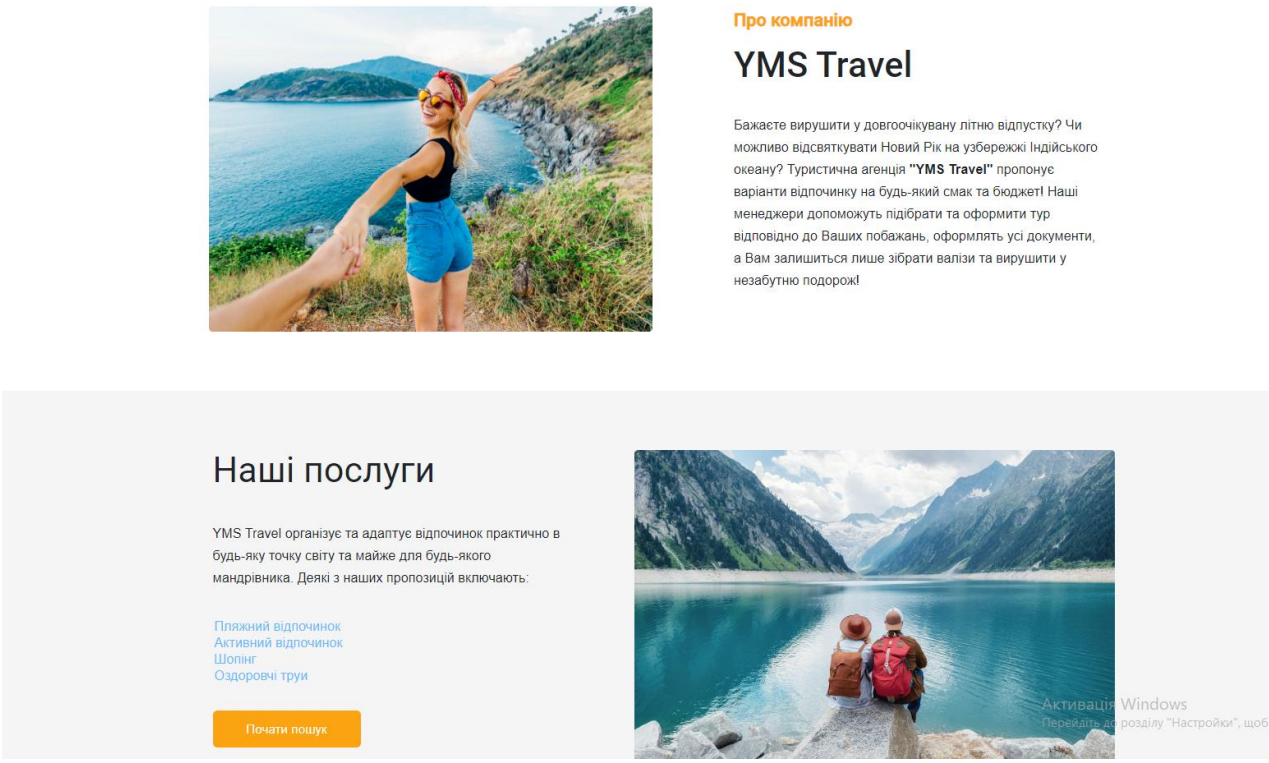


Рис. 4.4 Відображення сторінки «Про нас»

Перейшовши за посиланням «Тури» в навігаційній панелі або натиснувши кнопку «Почати пошук», користувач потрапляє на сторінку турів. На цій сторінці, в області header і footer, розташовані три основні блоки: головний фільтр, додатковий фільтр та блок для виведення основного контенту.

Важливо зазначити, що фільтри працюють незалежно один від одного, тобто користувач може застосувати лише один фільтр для виведення результатів, а не використовувати їх одночасно. Приклад роботи сторінки турів та фільтра «Звідки» показаний на рисунку 4.5

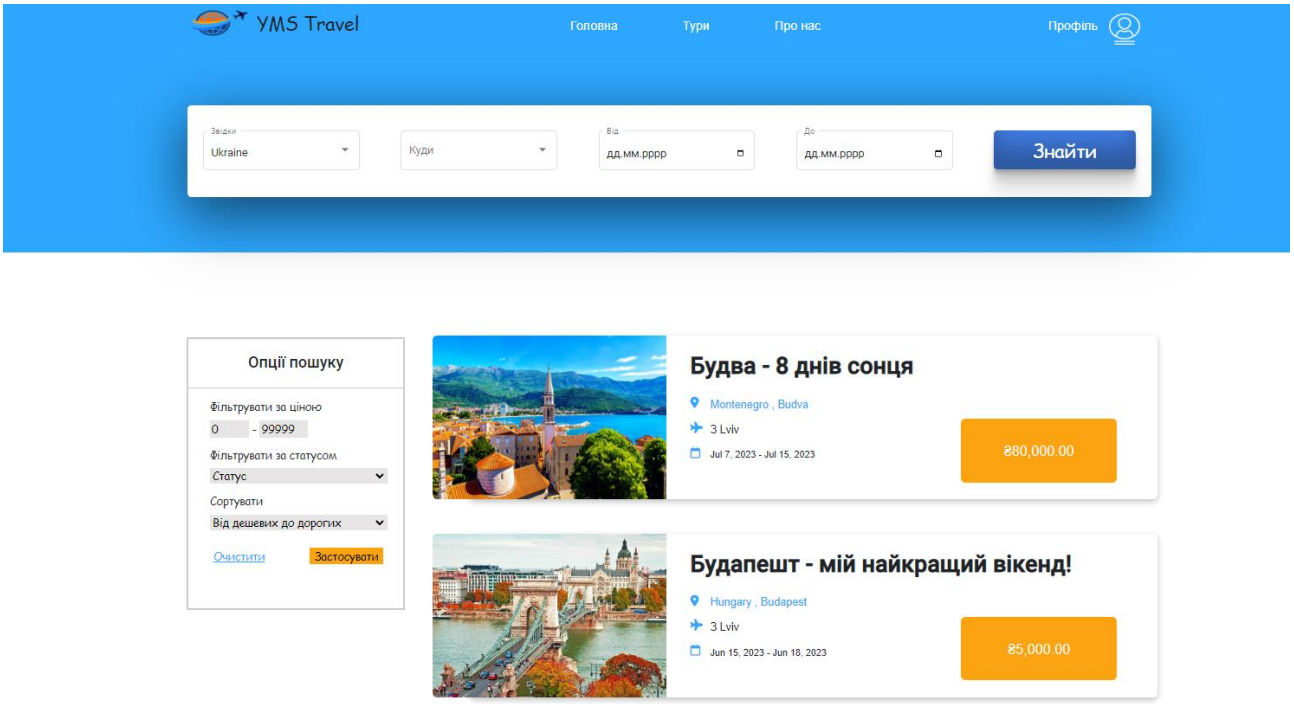


Рис. 4.5 Виведення турів з використанням фільтру «Звідки»

Крім того, ми можемо використовувати додаткові фільтри в лівому куті для сортування значень від найдорожчих до найдешевших. Варто зазначити, що в проекті реалізована пагінація, яка дозволяє переміщатися між сторінками та отримувати дані порціями. Приклад роботи сортування та додаткових фільтрів продемонстровано на рис. 4.6

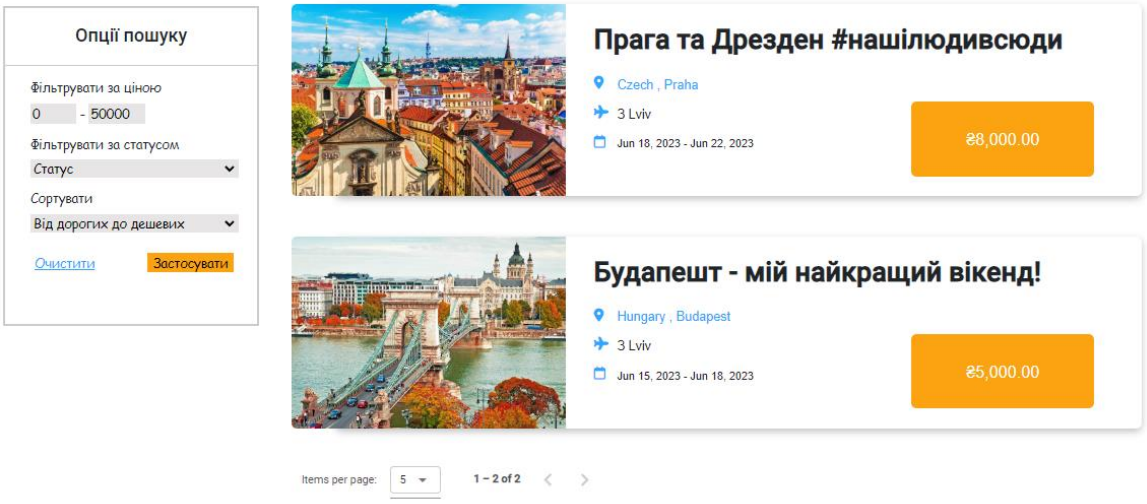


Рис. 4.6 Робота додаткових фільтрів та сортування

Переглядаючи тури, користувач має можливість перейти на сторінку конкретного туру та ознайомитися з детальною інформацією, натиснувши на кнопку, що відображає ціну туру. На сторінці конкретного туру, окрім інформації, зазначеної на сторінці списку турів, вказується кількість доступних місць, тип відпочинку, на скільки людей розрахований тур, а також, при наявності, знижка.

На рис. 4.7 зображено виведення окремої сторінки туру.

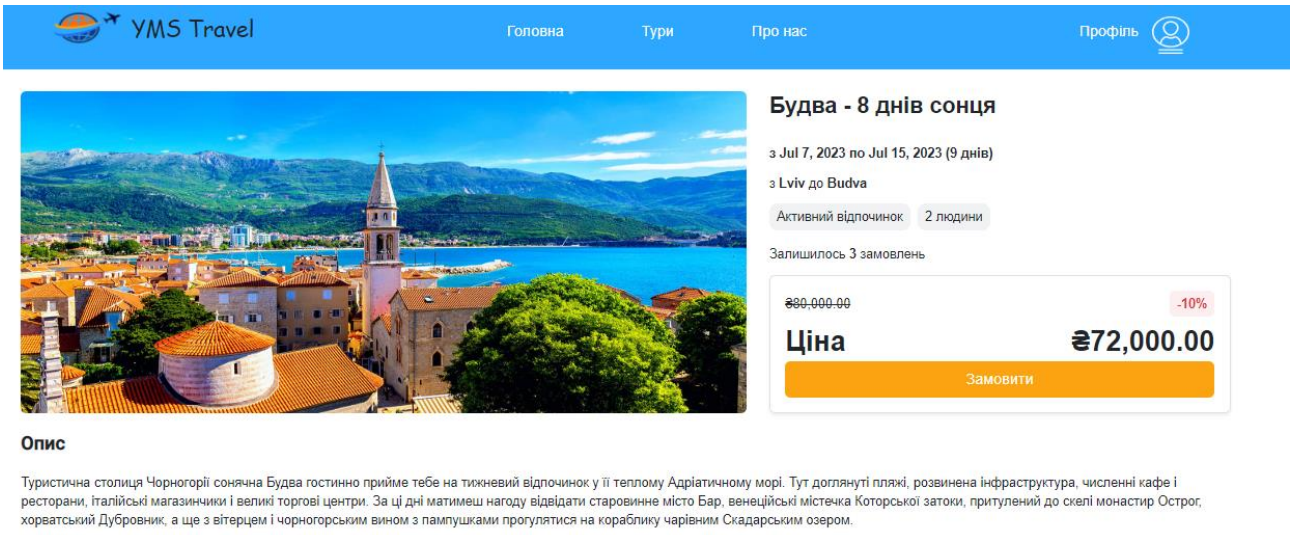


Рис. 4.7 Виведення окремого туру

Якщо тур сподобався користувачу, він може його замовити. Проте, якщо користувач не авторизований, він не зможе здійснити замовлення. У разі спроби замовити тур без авторизації, з’явиться повідомлення, що користувач повинен пройти реєстрацію. Приклад цього повідомлення показано на рис. 4.8

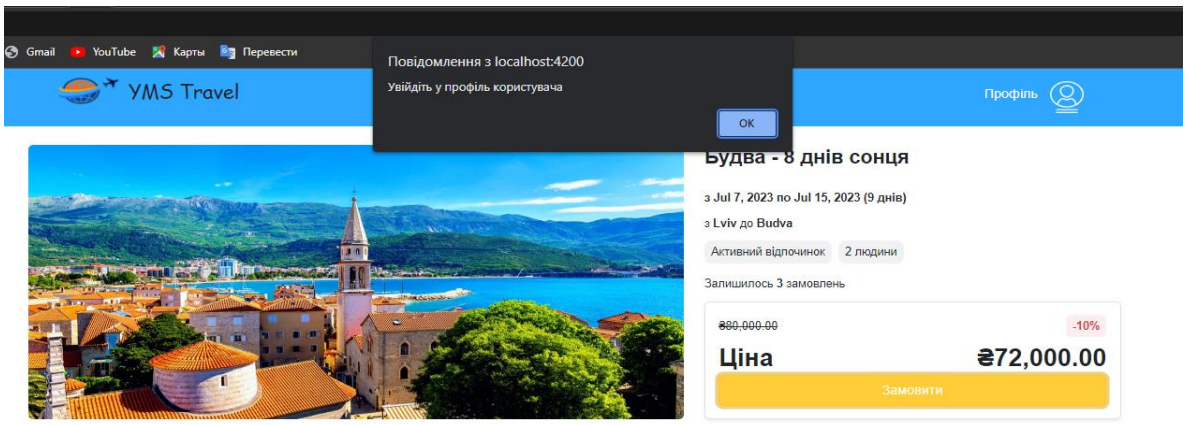


Рис. 4.8 Повідомлення про необхідність авторизуватись

Користувач може увійти в свій профіль, натиснувши на посилання «Профіль». Якщо користувач ще не зареєстрований, він буде перенаправлений на сторінку авторизації, де зможе або зареєструватися, якщо не має акаунту, або увійти за наданими даними. Приклад сторінки авторизації зображено на рис. 4.9

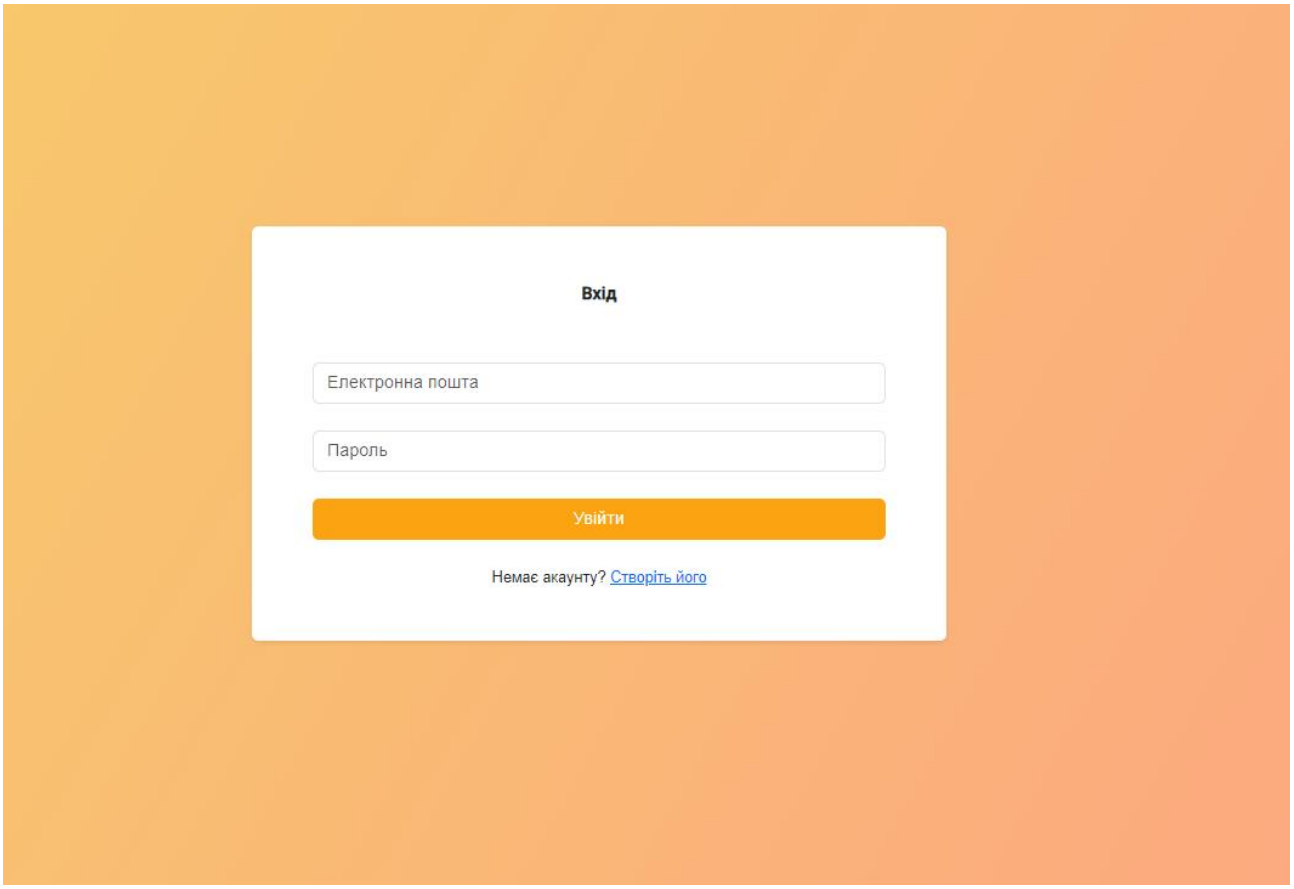


Рис. 4.9 Сторінка авторизації

Сторінка реєстрації на рис. 4.10

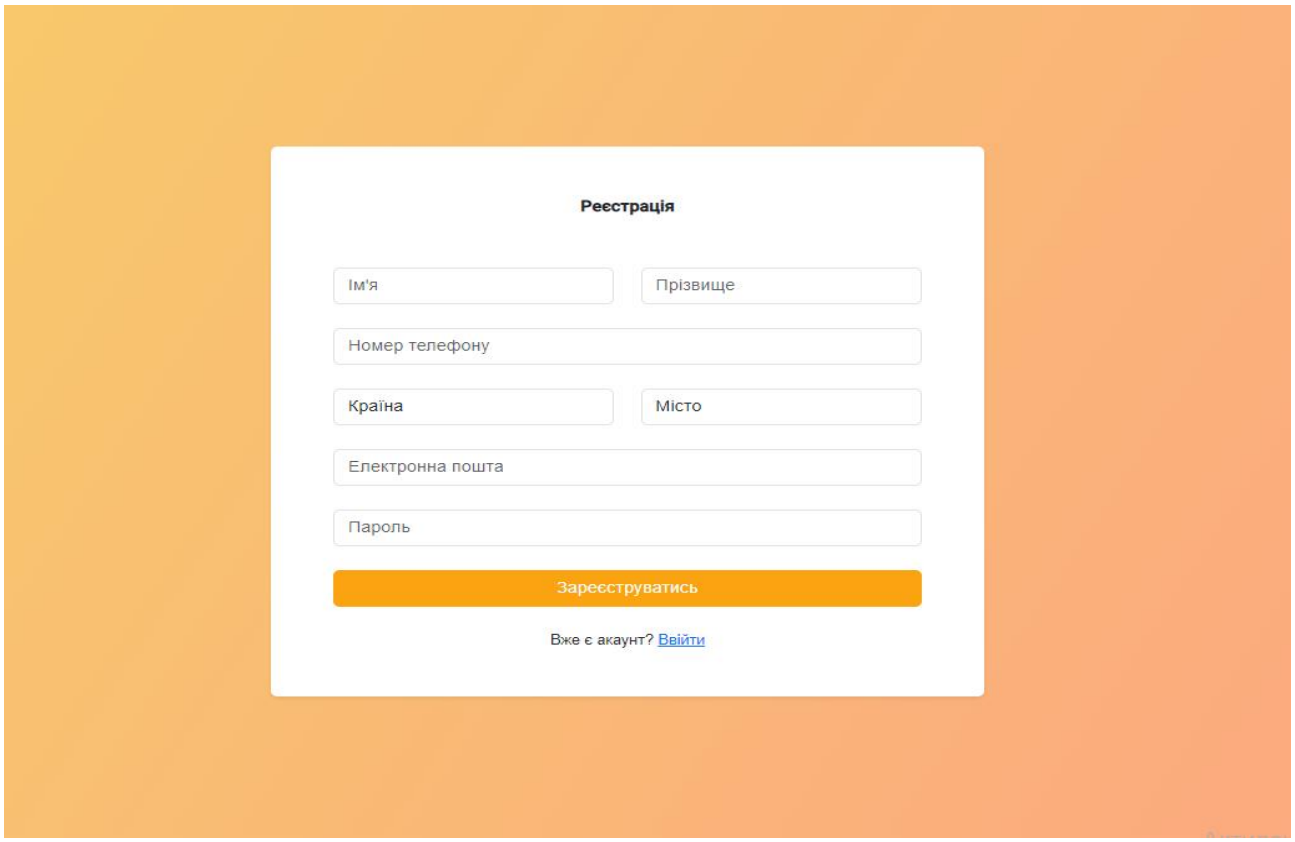


Рис. 4.10 Сторінка реєстрації

Після авторизації користувач потрапляє на головну сторінку свого профілю, де може переглянути свої особисті дані, а також має можливість їх редагувати. Крім того, на сторінці є кілька навігаційних панелей, одна з яких містить інформацію про замовлені користувачем тури. Приклад сторінки профілю користувача зображено на рис. 4.11

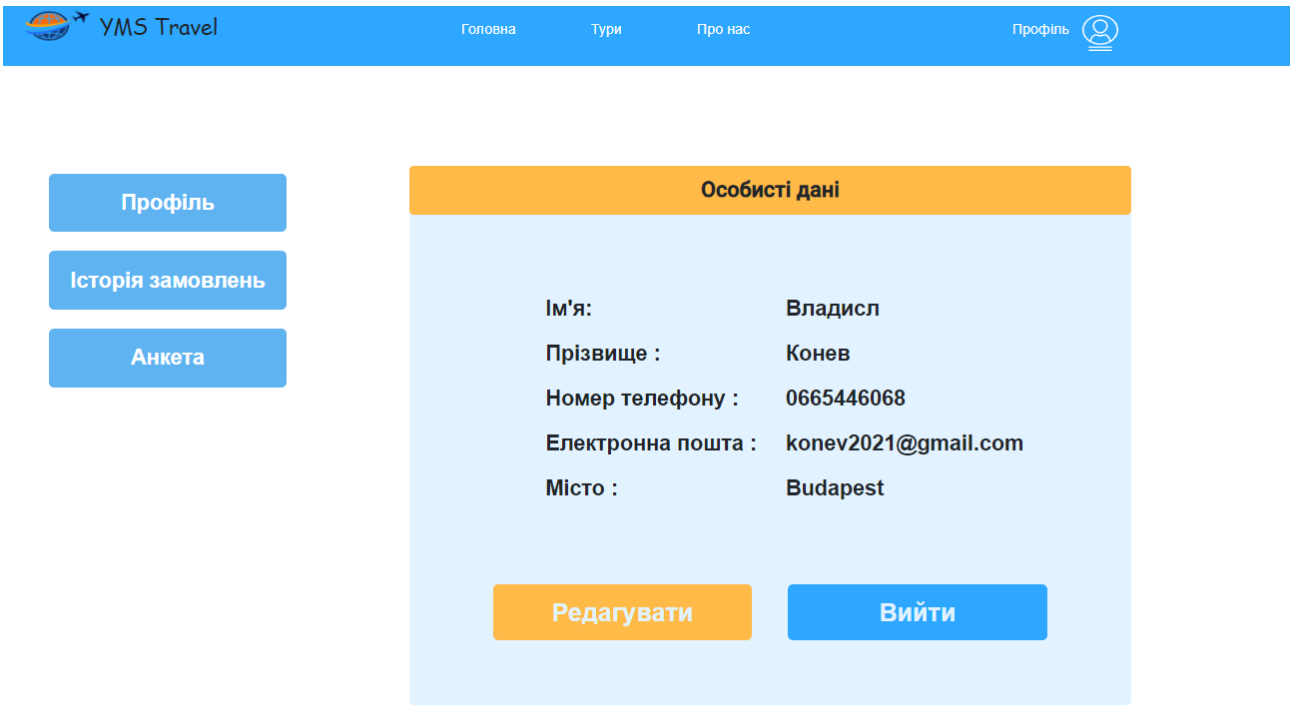


Рис. 4.11 Сторінка профіля користувача

Натискаючи на кнопку «Редагувати», користувач потрапляє на сторінку форми, де може змінити необхідні дані. Наприклад, змінити ім’я на «Владислав» і місто на «Київ», після чого натискає «Зберегти».

Приклад сторінки редагування інформації користувача зображено на рис. 4.12

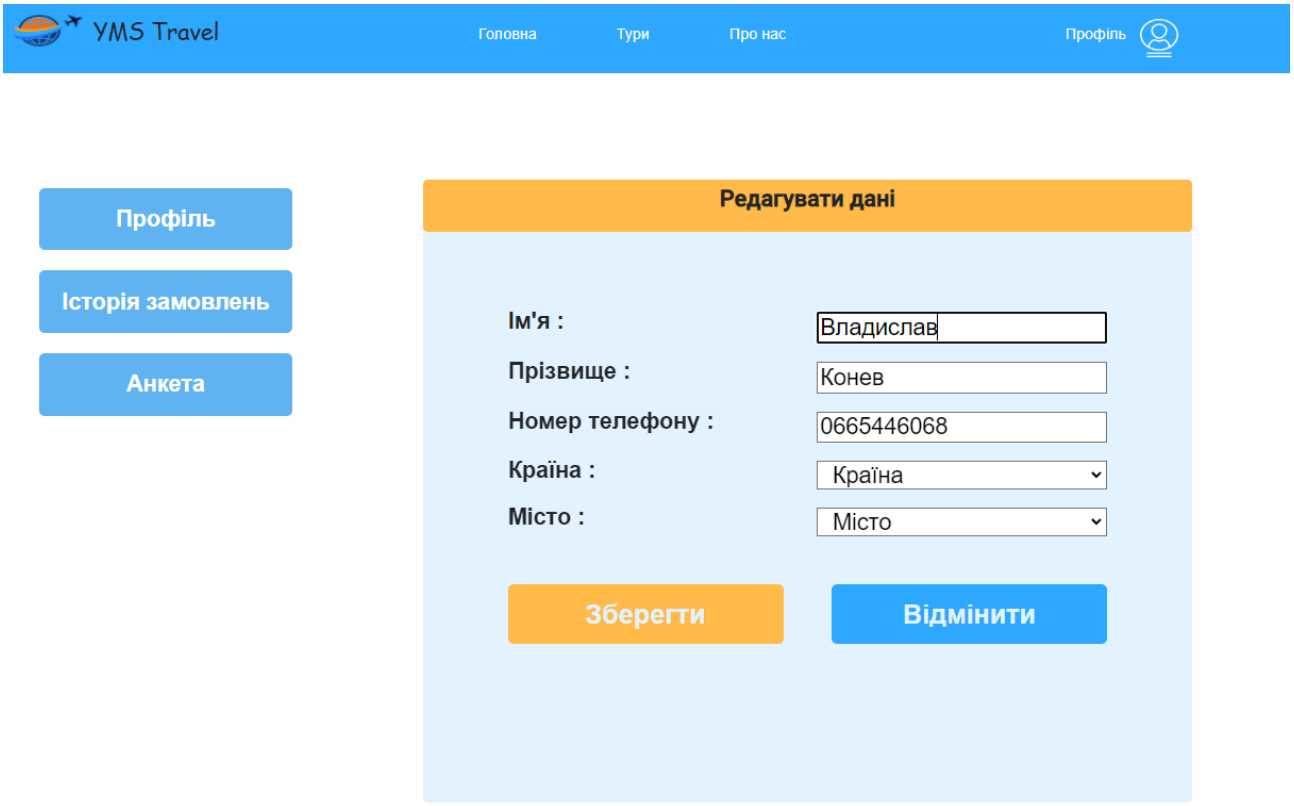


Рис. 4.12 Сторінка редагування даних користувача

На рис. 4.13 зображено сторінку профіля зі зміненими параметрами .

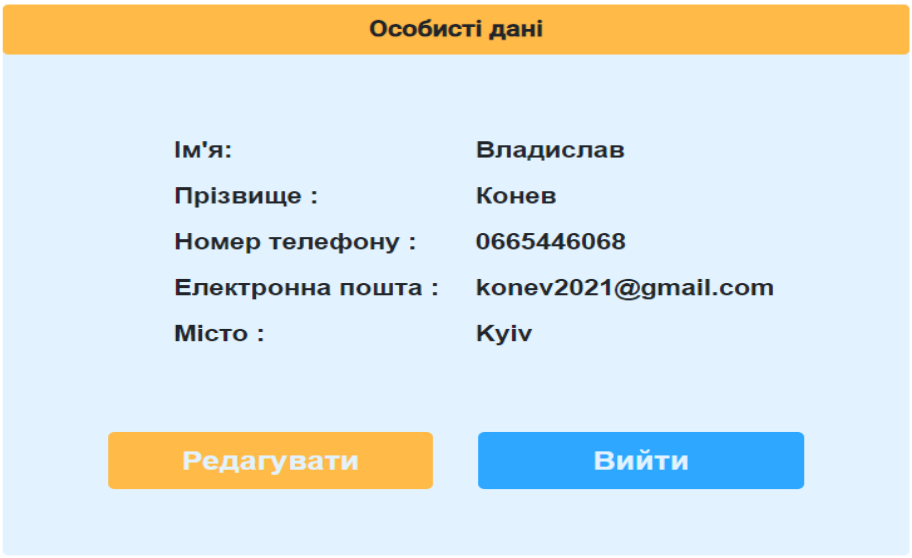


Рис. 4.13 Змінені дані у профілі користувача

Після авторизації користувач має змогу замовити тур. Натискаючи на кнопку «Замовити», тур додається до замовлень, і користувач автоматично переходить на сторінку замовлених турів. Варто зазначити кілька важливих моментів: по-перше, кількість доступних турів після замовлення зменшується; по-друге, при переході на сторінку замовлених турів користувач бачить тільки ті тури, які він замовив. Приклад сторінки замовлених турів зображено на рис. 4.14

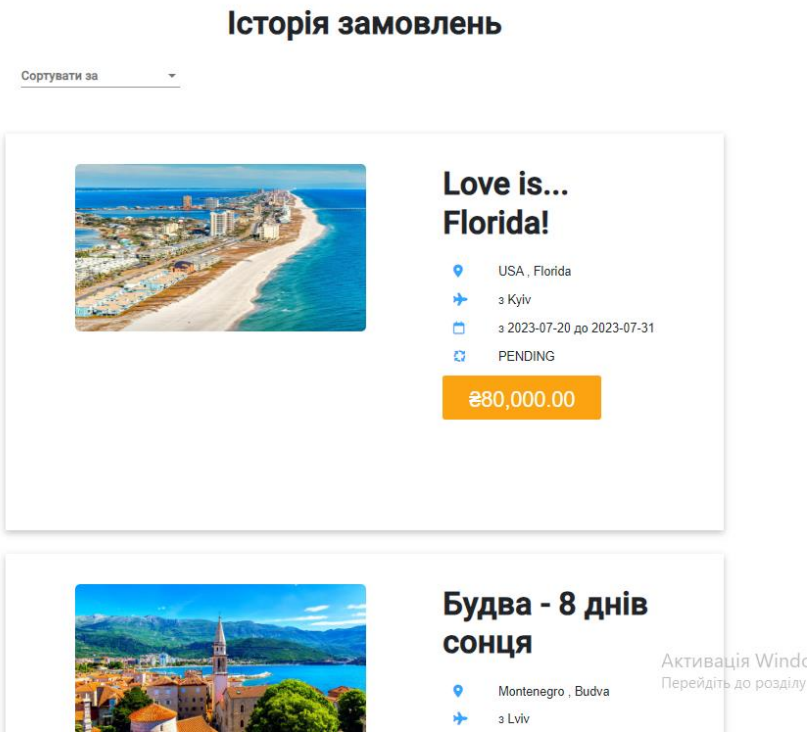


Рис. 4.14 Відображення сторінки замовлених турів

У системі передбачено три основні ролі: зареєстрований користувач, незареєстрований користувач та адміністратор, для кожної з яких реалізована відповідна логіка доступу.

Якщо користувач входить в систему як адміністратор, йому стає доступним окремий профіль. У профілі адміністратора є функції для перегляду історії замовлень, а також для перегляду всіх доступних турів. Адміністратор

має можливість оновлювати та видаляти тури. Для кожного з цих компонентів також реалізовано сортування для зручності роботи з даними.

На рис. 4.15 зображено сторінку усіх замовлених турів.

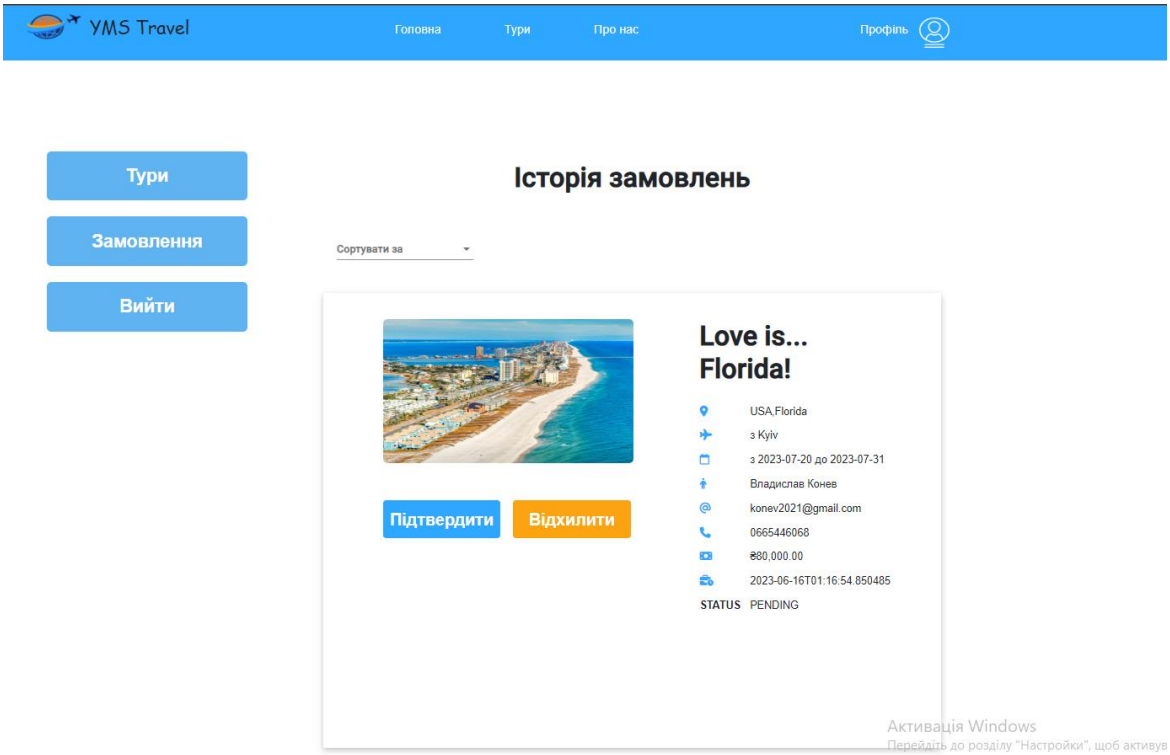


Рис. 4.15 Профіль адміністратора та сторінка історії замовлень

На рис. 4.16 зображено сторінку списку турів, доступних для перегляду.

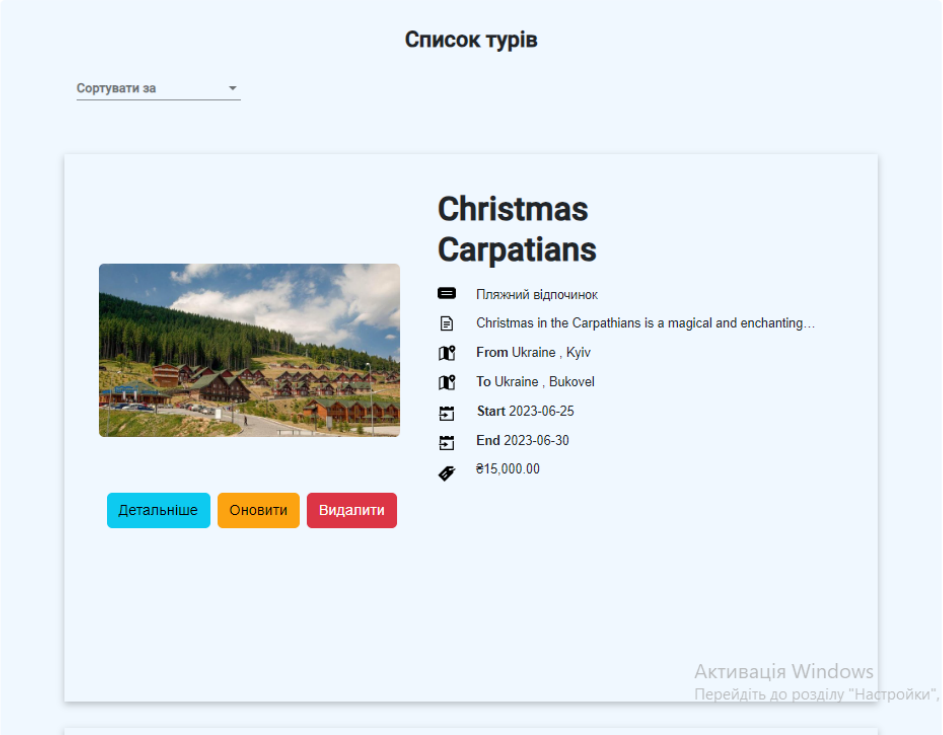


Рис. 4.16 Сторінка списку турів у панелі адміністратора

Передбачено виконання трьох функції з кожним туром:

1) переглянути детальну інформацію про тур(відкривається сторінка детальною інформацією про тур);

2) видалити тур(тур видаляється зі списку турів та з бази даних);

3) оновити тур(тур оновлюється у списку турів та у базі даних).

При оновленні туру, ми потрапляємо на сторінку форми редагування туру, де можемо задати відповідні значення, які будуть змінені після натискання клавіші «Зберегти».

На рис. 4.17 зображено сторінку редагування туру.

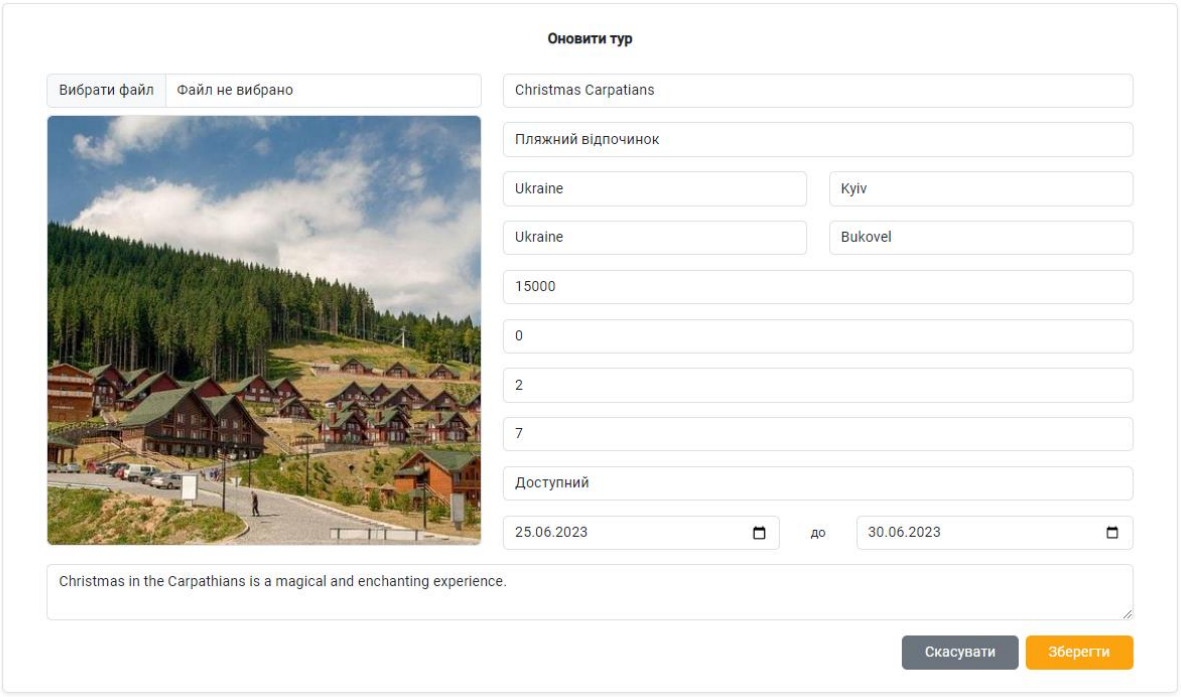


Рис. 4.17 Форма редагування туру

Для роботи з Telegram-ботом, який відправляє акти звірки для моніторингу розрахунків з клієнтом по туристичному агенству, необхідно:

1) перейти в месенджер Telegram та запустити бота, натиснувши на команду /start;

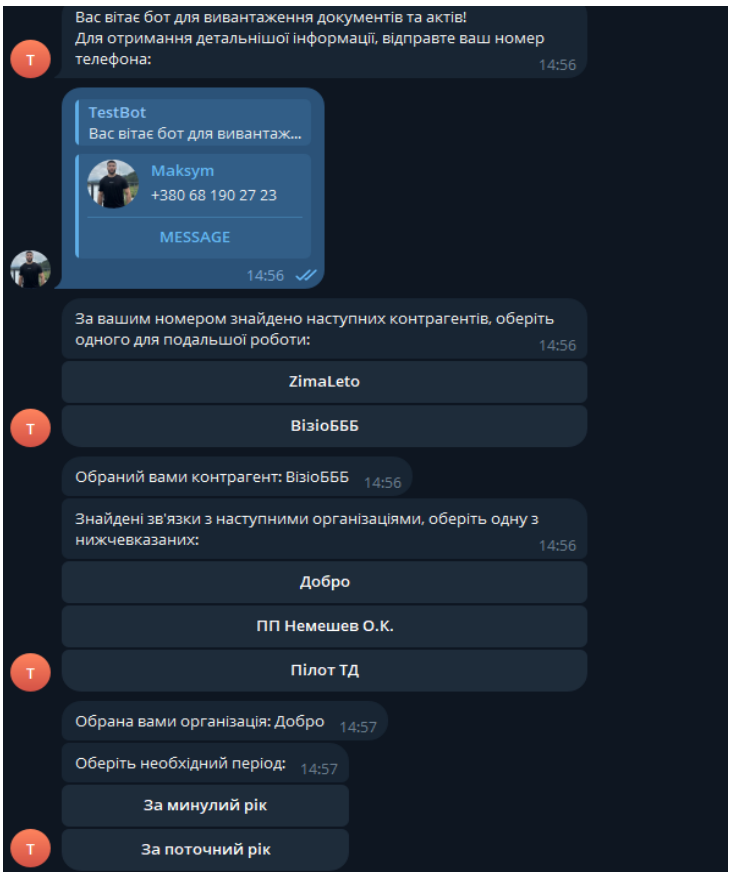
2) в ході роботи алгоритму, в списку контрагентів знаходиться контрагенти, в яких присутній контактний номер телефону, відправлений користувачем, виводиться інлайн-клавіші, на яких відображаються контрагенти, одного з них потрібно обрати, для пошуку пов’язаних організацій;

3) обираємо одну з організацій;

4) обираємо необхідний період;

5) ми отримаємо акт звірки, у форматі PDF якщо є певні розрахунки між обраними контрагентами та організаціями, або ж не отримуємо, якщо немає.

Приклад роботи телеграм-боту відображений на рис. 4.18



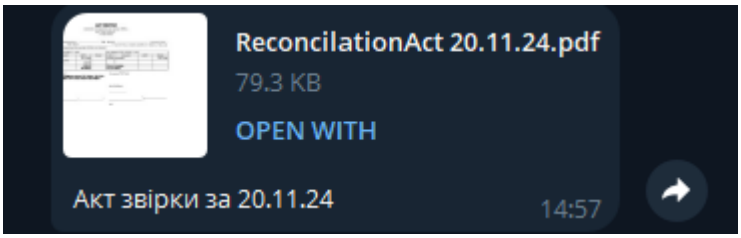


Рис. 4.18 Приклад роботи Телеграм-боту

ВИСНОВКИ

Розроблений програмний продукт є комплексною системою, яка об'єднує функціональність туристичного агентства з можливостями туристичного агрегатора та інтегрованою рекомендаційною системою. Програмний продукт охоплює широкий спектр задач, що стосуються туризму та туристичної діяльності, задовольняючи поставлені вимоги до кваліфікаційної роботи.

Незважаючи на складність деяких функцій, система є інтуїтивно зрозумілою та легкою у використанні. Використвння мови Java, фреймворків Spring Boot, Angular і Flask, системи контролю версій Git, а також мови Python, яка була використана для реалізації рекомендаційної системи дозволило комплексно реалізувати вищеописаний програмний продукт. Реалізація Telegram-бота сприяє кращій роботі з бізнес-клієнтами при роботі з фінансовою документацією.

Низькі вимоги до апаратного забезпечення дозволяють запускати програмний продукт на будь-якому пристрої, що забезпечує доступ до нього широкому колу користувачів. У порівнянні з аналогами, які надають лише функціональність туристичного агрегатора, розроблений продукт пропонує набагато ширший спектр послуг, включаючи можливості туристичного агентства та інтегровану рекомендаційну систему. Це значно підвищує попит на відповідні тури серед користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Афанасьєв І.Ю. Історія Туризму. – Київ: Зірка. – С.210-250.
2. Spring Boot документація. – [Електронний ресурс] Режим доступу: https://spring.io/projects/spring-boot.
3. Angular – [Електронний ресурс] Режим доступу: https://glyanec. net/ua/ blog/onlayn-chat-dlya-internet-mahazynu
4. Технічна документація «Python». – [Електронний ресурс] Режим доступу: https://docs.python.org/3/
5. The Hundred-page Machine Learning Book: Andriy Burkov. C. 220-260
6. Uml Diagram Types Guide. – [Електронний ресурс] Режим доступу: https://creately.com/blog/diagrams/uml-diagram-types-examples/
7. Spring Security. – [Електронний ресурс] Режим доступу: https:// spring.io/projects/spring-security
8. Simple SVD Algorithm. – [Електронний ресурс] Режим доступу: https://medium.com/@ebimsv/mastering-linear-algebra-part-8-singular-value-de-composition-svd-3dbc34c91908
9. Cosine Similary and TFIDF. – [Електронний ресурс] Режим доступу: https://goodboychan.github.io/python/datacamp/natural_language_processing/2020/07/17/04-TF-IDF-and-similarity-scores.html
10. Технічна документація «Flask». – [Електронний ресурс] Режим доступу: https://flask.palletsprojects.com/en/2.3.x/

ДОДАТКИ

Додаток А (Вебсайт – Back End)

A1. BackEnd/BackEndApplication.java

```
package com.yukon.travel;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.lang.NonNull;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
@SpringBootApplication
public class BackEndApplication {
	public static void main(String[] args) {
		SpringApplication.run(BackEndApplication.class, args);
	}
	@Bean
	public WebMvcConfigurer corsConfigurer() {
		return new WebMvcConfigurer() {
			@Value("${application.cors.origin}")
			private String frontendOrigin;
			@Override
			public void addCorsMappings(@NonNull CorsRegistry registry) {
				registry.addMapping("/api/auth/logout").allowedOrigins(frontendOrigin);}};}
}
```

A2. BackEnd/service/UserService

```
package com.yukon.travel.service;
import com.yukon.travel.entity.User;
import java.util.List;
public interface UserService {
   List<User> getAll();
   User get(Long id);
   User getByEmail(String email);
   User create(User user);
   User update(User user);
   void delete(User user);
}
```

A3. BackEnd/service/impl/UserServiceImpl

```
package com.yukon.travel.service.impl;
import com.yukon.travel.entity.User;
import com.yukon.travel.exception.DatabaseFetchException;
import com.yukon.travel.exception.DeleteException;
import com.yukon.travel.exception.DuplicateException;
import com.yukon.travel.repository.UserRepository;
import com.yukon.travel.service.UserService;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class UserServiceImpl implements UserService {
   private final UserRepository repository;
   private final PasswordEncoder passwordEncoder;
   public UserServiceImpl(UserRepository repository, PasswordEncoder passwordEncoder) {
      this.repository = repository;
      this.passwordEncoder = passwordEncoder;
   }
   @Override
   public List<User> getAll() {
```

```
        return repository.findAll();
    }
    @Override
    public User get(Long id) {
        return repository.findById(id).orElseThrow(()-> new DatabaseFetchException("Could not find User entity with id: " + id));
    }
    @Override
    public User getByEmail(String email) {
        return repository.findByEmail(email).orElseThrow(() -> new DatabaseFetchException("Could not find User entity with email " + email));
    }
    @Override
    public User create(User user) {
        if (repository.findIdByEmail(user.getEmail()) != null) {
            throw new DuplicateException("акаунт з ел. поштою '" + user.getEmail() + "' вже існує");
        }
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        return repository.save(user);
    }
    @Override
    public User update(User user) {
        Long emailId = repository.findIdByEmail(user.getEmail());
        if (emailId != null && !emailId.equals(user.getId())) {
            throw new DuplicateException("акаунт з ел. поштою '" + user.getEmail() + "' вже існує");
        }
        return repository.save(user);
    }
    @Override
    public void delete(User user) {
        try {
            repository.delete(user);
        } catch (DataIntegrityViolationException e) {
            throw new DeleteException("Could not delete User entity with id " + user.getId() + " because it is referenced in Token or Order");  } }}
```

A4. BackEnd/service/TourService

```
package com.yukon.travel.service;
import com.yukon.travel.dto.tour.PageTourResponse;
import com.yukon.travel.entity.Location;
import com.yukon.travel.entity.Tour;
import com.yukon.travel.entity.TourStatus;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import java.time.LocalDate;
import java.util.List;
public interface TourService {
    List<Tour> getAll();
     List<PageTourResponse> getAllToursPage(int pageNo,
                                            int pageSize,
                                            String sortBy,
                                            String sortDir,
                                            TourStatus status,
                                            int priceStart,
                                            int priceEnd,
                                            Location departure,
                                            Location destination,
                                            LocalDate startDate,
                                            LocalDate endDate);
    List<Tour> getAllAvailable();
    Tour get(Long id);
    Tour save(Tour tour);
    void delete(Long id);
```

```
    String getImagePathById(Long id);
    List<Tour> findToursByFilter(TourStatus status,int priceStart, int priceEnd);
}
```

A5. BackEnd/service/impl/TourServiceImpl

```
package com.yukon.travel.service.impl;
import com.yukon.travel.dto.tour.PageTourResponse;
import com.yukon.travel.dto.tour.TourResponse;
import com.yukon.travel.entity.Location;
import com.yukon.travel.entity.Tour;
import com.yukon.travel.entity.TourStatus;
import com.yukon.travel.exception.DatabaseFetchException;
import com.yukon.travel.exception.DeleteException;
import com.yukon.travel.mapper.TourMapper;
import com.yukon.travel.repository.TourRepository;
import com.yukon.travel.service.LocationService;
import com.yukon.travel.service.TourService;
import jakarta.persistence.criteria.Predicate;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Sort;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;
import org.springframework.data.domain.Pageable;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import static org.springframework.data.domain.Sort.by;
@Service
public class TourServiceImpl implements TourService {
    private final TourRepository repository;
    private final TourMapper mapper;
    private final LocationService locationService;
    public TourServiceImpl(TourRepository repository, TourMapper mapper,LocationService locationService) {
        this.repository = repository;
        this.locationService = locationService;
        this.mapper = mapper;
    }
    @Override
    public List<Tour> getAll() {
        return repository.findAll();
    }
    public List<PageTourResponse> getAllToursPage(int pageNo,
                                                  int pageSize,
                                                  String sortBy,
                                                  String sortDir,
                                                  TourStatus status,
                                                  int priceStart,
                                                  int priceEnd,
                                                  Location departure,
                                                  Location destination,
                                                  LocalDate startDate,
                                                  LocalDate endDate) {
        Sort sort = sortDir.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortBy).ascending()
                : Sort.by(sortBy).descending();
        Pageable pageable = PageRequest.of(pageNo, pageSize, sort);
        Specification<Tour> specification = (root,query,criteriaBuilder) -> {
            List<Predicate> predicates = new ArrayList<>();
            if(status != null) {
                predicates.add(criteriaBuilder.equal(root.get("status"),status));
            }
predicates.add(criteriaBuilder.greaterThan(root.get("price"),priceStart));
```

```
predicates.add(criteriaBuilder.lessThan(root.get("price"),priceEnd));

            if(departure != null) {
     predicates.add(criteriaBuilder.equal(root.get("departure"), departure));
            }
            if(destination != null) {
          predicates.add(criteriaBuilder.equal(root.get("destination"), destination));
            }
            if(startDate != null) {
predicates.add(criteriaBuilder.greaterThan(root.get("startDate"), startDate));
            }
            if(endDate != null) {
          predicates.add(criteriaBuilder.lessThan(root.get("endDate"), endDate));
            }
          Return criteriaBuilder.and(predicates.toArray(new Predicate[0]));
       };
       Page<Tour> tours = repository.findAll(specification,pageable);

       List<Tour> listOfTours = tours.getContent();

       List<PageTourResponse> responseList = new ArrayList<>();

       for (Tour tour : listOfTours) {
          PageTourResponse pageTourResponse = new PageTourResponse();
          pageTourResponse.setContent(Collections.singletonList(mapper.toResponse(tour)));
          pageTourResponse.setPageNo(tours.getNumber());
          pageTourResponse.setPageSize(tours.getSize());
          pageTourResponse.setTotalElements(tours.getTotalElements());
          pageTourResponse.setTotalPages(tours.getTotalPages());
          pageTourResponse.setLast(tours.isLast());

          responseList.add(pageTourResponse);
       }
       return responseList;
    }
    @Override
    public List<Tour> getAllAvailable() {
       return repository.findAllAvailable();
    }
    @Override
    public Tour get(Long id) {
       return repository.findById(id).orElseThrow(() -> new DatabaseFetchException("Could not find Tour entity with
id " + id));
    }
    @Override
    public Tour save(Tour tour) {
       return repository.save(tour);
    }
    @Override
    public void delete(Long id) {
       if (!repository.existsById(id)) {
          throw new DatabaseFetchException("Could not find Tour entity with id " + id);
       }
       try {
          repository.deleteById(id);
       } catch (DataIntegrityViolationException e) {
          throw new DeleteException("Could not delete Tour entity with id " + id + " because it is already ordered");
       }
    }
    @Override
    public String getImagePathById(Long id) {
       return repository.findImagePathById(id);
    }
```

```
    @Override
    public List<Tour> findToursByFilter(TourStatus status, int priceStart, int priceEnd) {
      return repository.findToursByFilter(status,priceStart,priceEnd);
    }
}
```

A6. BackEnd/service/OrderService

```
package com.yukon.travel.service;
import com.yukon.travel.dto.order.PageOrderResponse;
import com.yukon.travel.entity.Order;
import java.util.List;
public interface OrderService {
    Order get(Long id);
    List<PageOrderResponse> getAllOrdersPage(int pageNo, int pageSize, String sortBy, String sortDir);
    List<Order> getAll();
    Order create(Order order);
    Order update(Order order);
    void delete(Order order);
    List<PageOrderResponse> getAllOrdersPageByUserId(Long userId,int pageNo, int pageSize, String sortBy, String
sortDir);
}
```

A7. BackEnd/service/impl/OrderServiceImpl

```
package com.yukon.travel.service.impl;
import com.yukon.travel.dto.order.PageOrderResponse;
import com.yukon.travel.dto.tour.PageTourResponse;
import com.yukon.travel.entity.Order;
import com.yukon.travel.entity.Tour;
import com.yukon.travel.entity.TourStatus;
import com.yukon.travel.exception.DatabaseFetchException;
import com.yukon.travel.exception.TourUnavailableException;
import com.yukon.travel.mapper.OrderMapper;
import com.yukon.travel.mapper.TourMapper;
import com.yukon.travel.repository.OrderRepository;
import com.yukon.travel.service.OrderService;
import com.yukon.travel.service.TourService;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
@Service
public class OrderServiceImpl implements OrderService {
    private final OrderRepository repository;
    private final TourService tourService;
    private final OrderMapper mapper;
    public OrderServiceImpl(OrderRepository repository, TourService tourService, OrderMapper mapper) {
        this.repository = repository;
        this.tourService = tourService;
        this.mapper = mapper;
    }
    @Override
    public Order get(Long id) {
        return repository.findById(id).orElseThrow(() -> new DatabaseFetchException("Could not find Order entity
with id " + id));
    }
    @Override
    public List<PageOrderResponse> getAllOrdersPage(int pageNo, int pageSize, String sortBy, String sortDir) {
        Sort sort = sortDir.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortBy).ascending()
                : Sort.by(sortBy).descending();
       Pageable pageable = PageRequest.of(pageNo,pageSize, sort);
       Page<Order> orders = repository.findAll(pageable);
```

```
        List<Order> orderList = orders.getContent();
        List<PageOrderResponse> responseList = new ArrayList<>();
        for (Order order : orderList) {
            PageOrderResponse pageOrderResponse = new PageOrderResponse();
            pageOrderResponse.setContent(Collections.singletonList(mapper.toResponse(order)));
            pageOrderResponse.setPageNo(orders.getNumber());
            pageOrderResponse.setPageSize(orders.getSize());
            pageOrderResponse.setTotalElements(orders.getTotalElements());
            pageOrderResponse.setTotalPages(orders.getTotalPages());
            pageOrderResponse.setLast(orders.isLast());
            responseList.add(pageOrderResponse);
        }
        return responseList;
    }
    @Override
    public List<Order> getAll() {
        return repository.findAll();
    }
    @Override
    public Order create(Order order) {
        Tour tour = order.getTour();
        if (tour.getStatus() == TourStatus.NOT_AVAILABLE) {
            throw new TourUnavailableException("Tour with id " + tour.getId() + " is not available for ordering");
        }
        Integer tourOrderCount = tour.getAvailableOrderCount();
        if (tourOrderCount == 0) {
            throw new TourUnavailableException("Tour with id " + tour.getId() + " is fully booked");
        }
        tour.setAvailableOrderCount(tourOrderCount - 1);
        tourService.save(tour);
        return repository.save(order);
    }
    @Override
    public Order update(Order order) {
        return repository.save(order);
    }
    @Override
    public void delete(Order order) {
        repository.delete(order);
    }
    @Override
    public List<PageOrderResponse> getAllOrdersPageByUserId(Long userId, int pageNo, int pageSize, String
sortBy, String sortDir) {
        Sort sort = sortDir.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortBy).ascending()
                : Sort.by(sortBy).descending();
        Pageable pageable = PageRequest.of(pageNo,pageSize, sort);

        Page<Order> orders = repository.findByUserId(userId,pageable);

        List<Order> orderList = orders.getContent();

        List<PageOrderResponse> responseList = new ArrayList<>();
        for (Order order : orderList) {
            PageOrderResponse pageOrderResponse = new PageOrderResponse();
            pageOrderResponse.setContent(Collections.singletonList(mapper.toResponse(order)));
            pageOrderResponse.setPageNo(orders.getNumber());
            pageOrderResponse.setPageSize(orders.getSize());
            pageOrderResponse.setTotalElements(orders.getTotalElements());
            pageOrderResponse.setTotalPages(orders.getTotalPages());
            pageOrderResponse.setLast(orders.isLast());

            responseList.add(pageOrderResponse);
        }
```

```
        return responseList;
    }
}
```

A10. BackEnd/security/JwtService

```
package com.yukon.travel.security;
import com.yukon.travel.service.TokenService;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.MalformedJwtException;
import io.jsonwebtoken.ExpiredJwtException;
import io.jsonwebtoken.UnsupportedJwtException;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
import io.jsonwebtoken.security.SignatureException;
import jakarta.servlet.http.HttpServletRequest;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Service;
import org.springframework.util.StringUtils;
import java.security.Key;
import java.util.Date;
@Service
public class JwtService {
    private static final Logger LOGGER = LoggerFactory.getLogger(JwtService.class);
    private final String secret;
    private final int expirationMs;
    private final TokenService tokenService;
    public JwtService(
            @Value("${application.security.jwt-secret}") String secret,
            @Value("${application.security.jwt-expiration-ms}") int expirationMs,
            TokenService tokenService
    ) {
        this.secret = secret;
        this.expirationMs = expirationMs;
        this.tokenService = tokenService;
    }
```

```
    public String parseJwt(HttpServletRequest request) {
        String headerAuth = request.getHeader("Authorization");
        if (StringUtils.hasText(headerAuth) && headerAuth.startsWith("Bearer ")) {
            return headerAuth.substring(7);
        }
        return null;
    }
    public String generateToken(Authentication authentication) {
        UserDetailsImpl userPrincipal = (UserDetailsImpl) authentication.getPrincipal();
        return Jwts.builder()
                .setSubject(userPrincipal.getUsername())
                .setIssuedAt(new Date())
                .setExpiration(new Date(new Date().getTime() + expirationMs))
                .signWith(getSigningKey(), SignatureAlgorithm.HS512)
                .compact();
    }
    public String getUsernameFromToken(String jwt) {
        return Jwts.parserBuilder()
                .setSigningKey(getSigningKey())
                .build()
                .parseClaimsJws(jwt)
                .getBody()
                .getSubject();
    }
    public boolean isTokenValid(String jwt) {
        try {
Jwts.parserBuilder().setSigningKey(getSigningKey()).build().parseClaimsJws(jwt);
            return true;
        } catch (MalformedJwtException e) {
            LOGGER.error("Invalid JWT token: {}", e.getMessage());
        } catch (ExpiredJwtException e) {
            LOGGER.error("JWT token is expired: {}", e.getMessage());
            tokenService.expireToken(jwt);
        } catch (UnsupportedJwtException e) {
            LOGGER.error("JWT token is unsupported: {}", e.getMessage());
        } catch (SignatureException e) {
            LOGGER.error("Invalid JWT signature: {}", e.getMessage());
```

```
        } catch (IllegalArgumentException e) {
            LOGGER.error("JWT claims string is empty: {}", e.getMessage());
        }
        return false;
    }
    private Key getSigningKey() {
        byte[] keyBytes = Decoders.BASE64.decode(secret);
        return Keys.hmacShaKeyFor(keyBytes);
    }
}
```

Додаток Б (Вебсайт – Front End)

1Б. FrontEnd/common/auth.model/auth-request.ts

```
export class AuthRequest {
    constructor(public email: string, public password: string) {}
}
```

2Б. FrontEnd/common/country.model/country.ts

```
export class Country {
  constructor(country: Country | null = null) {
    this.id = country?.id ?? '';
    this.name = country?.name ?? '';
  }
  id: string;
  name: string;
}
```

3Б. FrontEnd/common/tour.model/tour.ts

```
import {Category} from "../tourCategory.model/category";
import {Location} from "../location.model/location";
export class Tour {
  constructor(tour: Tour | null = null) {
    this.id = tour?.id ?? '';
    this.category = tour?.category ?? new Category();
    this.name = tour?.name ?? '';
    this.description = tour?.description ?? '';
    this.imagePath = tour?.imagePath ?? '';
    this.departure = tour?.departure ?? new Location();
    this.destination = tour?.destination ?? new Location();
```

```
    this.initialPrice = tour?.initialPrice ?? 0;
    this.discountPercentage = tour?.discountPercentage ?? 0;
    this.numOfPeople = tour?.numOfPeople ?? 0;
    this.availableOrderCount = tour?.availableOrderCount ?? 0;
    this.status = tour?.status ?? '';
    this.startDate = tour?.startDate ?? new Date();
    this.endDate = tour?.endDate ?? new Date();
    this.createdAt = tour?.createdAt ?? new Date();
  }
  id: string;
  category: Category;
  name: string;
  description: string;
  imagePath: string;
  departure: Location;
  destination: Location;
  initialPrice: number;
  discountPercentage: number;
  numOfPeople:number;
  availableOrderCount: number;
  status: string;
  startDate: Date;
  endDate: Date;
  createdAt: Date;
}
```

Додаток В (Рекомендаційна система)

В1. RecommendSystem/flask/app.py

```
 from sqlalchemy import create_engine
import pymysql
import pandas as pd
db_connection_str  = 'mysql+pymysql://ymsteam:' + 'team123456' + '@localhost:3306/touragency'
db_connection = create_engine(db_connection_str)

df = pd.read_sql('SELECT * FROM tour', con = db_connection)
df.to_dict()
connect = db_connection.connect()
frame = pd.read_sql("select * from tour", connect);
```

```
pd.set_option('display.expand_frame_repr', False)
print(frame)
connect.close()
frame.head(10)
ratings_data = pd.read_csv('rating.csv')
ratings_data.head(10)
from surprise import Dataset
from surprise import Reader
import surprise
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(ratings_data[['place_id', 'user_id', 'rating']], reader)
from surprise import SVD
from surprise.model_selection import cross_validate
svd = SVD(verbose=True, n_epochs=10)
cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=3, verbose=True)
import random
import difflib
def get_tour_id(tour_title, metadata):
    existing_titles = list(metadata['name'].values)
    closest_titles = difflib.get_close_matches(tour_title, existing_titles)
    book_id = metadata[metadata['name'] == closest_titles[0]]['id'].values[0]
    return book_id
def get_tour_info(tour_id, metadata):
    tour_info = metadata[metadata['id'] == tour_id][['id', 'name',
                                        'description', 'price']]
    return tour_info.to_dict(orient='records')
def predict_review(user_id, tour_title, model, metadata):
    tour_id = get_tour_id(tour_title, metadata)
    review_prediction = model.predict(uid=user_id, iid=tour_id)
    return review_prediction.est
def generate_recommendation(user_id, model, metadata, thresh=4):
    tour_titles = list(metadata['name'].values)
    random.shuffle(tour_titles)
    for tour_title in tour_titles:
        rating = predict_review(user_id, tour_title, model, metadata)
        if rating >= thresh:
            tour_id = get_tour_id(tour_title, metadata)
```

```
        return get_tour_info(tour_id, metadata)
generate_recommendation(2, svd, frame)
```

Telegram Bot (BAF)

A. Спільний модуль

```
#Область ПриемЗапроса
Функция УдалитьСимволы(ИсходнаяСтрока, МассивСимволовДляУдаления)
    Для Каждого Символ Из МассивСимволовДляУдаления Цикл
            ИсходнаяСтрока = СтрЗаменить(ИсходнаяСтрока,Символ, "");
    КонецЦикла;
    Возврат ИсходнаяСтрока;
КонецФункции
Функция ПолучитьТекстЗапросаКТелеграмм(ИмяКоманды, СтруктураПараметров)
     Настройки = РегистрыСведений.ТГ_Настройки.СоздатьМенеджерЗаписи();
     Настройки.НалаштуванняМодуля = "ТГ_TelegramBot";
     Настройки.Прочитать();
    Сервер = "api.telegram.org";
    Адрес = "bot<token>/";
    Адрес = СтрЗаменить(Адрес, "<token>", Настройки.ТокенТелеграм);
    Адрес = Адрес + ИмяКоманды;
    ТекстПараметров = СформироватьТекстЗапроса(СтруктураПараметров);
    Если НЕ ПустаяСтрока(ТекстПараметров) Тогда
            ТекстПараметров = "?" + ТекстПараметров;
            Адрес = Адрес + ТекстПараметров;
    КонецЕсли;
    Возврат Адрес;
КонецФункции
Функция СформироватьТекстЗапроса(СтруктураПараметров)
    Если СтруктураПараметров = Неопределено Тогда
            Возврат "";
    КонецЕсли;
    ТекстЗапроса = "";

    Для Каждого ЭлементСтруктуры Из СтруктураПараметров Цикл
            Если ЭлементСтруктуры.Значение = Неопределено Тогда
                Продолжить;
            КонецЕсли;
```

```
            Если ТипЗнч(ЭлементСтруктуры.Значение) = Тип("Структура") Тогда
                ТекстЗапроса = ТекстЗапроса + СформироватьТекстЗапроса(ЭлементСтруктуры.Значение);
                Продолжить;
            КонецЕсли;
            ТекстЗапроса = ТекстЗапроса + ?(ПустаяСтрока(ТекстЗапроса), "", "&") + ЭлементСтруктуры.Ключ + "=" + ЭлементСтруктуры.Значение;
    КонецЦикла;
    Возврат ТекстЗапроса;
КонецФункции

Функция ОтправитьСообщениеАдминистратору(chat_id)
    НомерТелефона = "";
    Запрос = Новый Запрос;
    Запрос.Текст =
    "ВЫБРАТЬ
    |   ТГ_ПользователиТелеграммПроходящиеРегистрацию.idПользователя КАК idПользователя,
    |   ТГ_ПользователиТелеграммПроходящиеРегистрацию.НомерТелефона КАК НомерТелефона
    |ИЗ
    |   РегистрСведений.ТГ_ПользователиТелеграммПроходящиеРегистрацию КАК ТГ_ПользователиТелеграммПроходящиеРегистрацию
    |ГДЕ
    |   ТГ_ПользователиТелеграммПроходящиеРегистрацию.idПользователя = &idПользователя";

    Запрос.УстановитьПараметр("idПользователя", chat_id);
    РезультатЗапроса = Запрос.Выполнить();
    ВыборкаДетальныеЗаписи = РезультатЗапроса.Выбрать();
    Пока ВыборкаДетальныеЗаписи.Следующий() Цикл
        НомерТелефона = ВыборкаДетальныеЗаписи.НомерТелефона;
    КонецЦикла;
    //}}КОНСТРУКТОР_ЗАПРОСА_С_ОБРАБОТКОЙ_РЕЗУЛЬТАТА
     Настройки = РегистрыСведений.ТГ_Настройки.СоздатьМенеджерЗаписи();
     Настройки.НалаштуванняМодуля = "ТГ_TelegramBot";
     Настрки.Прочитать();
    Если Настройки.ТокенТелеграм  <> "" И Настройки.АйдиАдминистратор <> 0 Тогда
        text = "Невдала спроба реєстрації в боті. Номер телефона - "  + НомерТелефона;
        Сервер = "api.telegram.org";
        Команда = "sendMessage";
```

```
    СтруктураТелаСообщения.Вставить("chat_id",Формат(Настройки.АйдиАдминистратор,"ЧГ="));
        СтруктураТелаСообщения.Вставить("text", text);
        ЗаписьJSON = Новый ЗаписьJSON;
        ЗаписьJSON.УстановитьСтроку();
        ЗаписатьJSON(ЗаписьJSON, СтруктураТелаСообщения);
        ТелоСообщения =  СтрЗаменить(ЗаписьJSON.Закрыть(),"\\","\");
        Адрес = ПолучитьТекстЗапросаКТелеграмм(Команда, Неопределено);
        Соединение = Новый HTTPСоединение(Сервер,,,,Новый ИнтернетПрокси(истина),5, Новый ЗащищенноеСоединениеOpenSSL);
        Запрос = Новый HTTPЗапрос(Адрес);
        Запрос.Заголовки.Вставить("Content-Type","application/json");
        Запрос.УстановитьТелоИзСтроки(ТелоСообщения, "CESU-8");
        Ответ = Соединение.ОтправитьДляОбработки(Запрос);
        Чтение = Новый ЧтениеJSON;
        Чтение.УстановитьСтроку(Ответ.ПолучитьТелоКакСтроку());
        ОтветСоотв = ПрочитатьJSON(Чтение,Истина);
        ok = ОтветСоотв["ok"] ;
    КонецЕсли
КонецФункции
Функция ОтправитьHTTPЗапрос(Метод,ПараметрыМетода = Неопределено,Данные)
    НаборЗаписейНастроек = РегистрыСведений.ТГ_Настройки.СоздатьМенеджерЗаписи();
    НаборЗаписейНастроек.НалаштуванняМодуля = "ТГ_TelegramBot";
    НаборЗаписейНастроек.Прочитать();
    Если НаборЗаписейНастроек.Выбран() Тогда
        ТокенДоступу = НаборЗаписейНастроек.ТокенТелеграм;
    КонецЕсли;
    Соединение = Новый HTTPСоединение("api.telegram.org",443,,,,Новый ЗащищенноеСоединениеOpenSSL(),);
    ЗапросHTTP = Новый HTTPЗапрос;
    Если Данные = Неопределено Тогда
        ЗапросHTTP.Заголовки.Вставить("Content-type","application/json");
    Иначе
        ЗапросHTTP.Заголовки.Вставить("Content-type", "multipart/form-data; boundary=" + Данные["Boundary"]);
        ТекстЗапроса = СформироватьТекстЗапросаДляДокумента(Данные);
        ЗапросHTTP.УстановитьТелоИзСтроки(ТекстЗапроса, КодировкаТекста.ANSI, ИспользованиеByteOrderMark.НеИспользовать);
    КонецЕсли;
    Если ПараметрыМетода <> Неопределено Тогда
```

```
            СтрокаПараметр = "";
            Для Каждого Параметр Из ПараметрыМетода Цикл
                СтрокаПараметр = СтрокаПараметр + Параметр;
            КонецЦикла;
            ЗапросHTTP.АдресРесурса = "bot" + ТокенДоступу + "/" + Метод + СтрокаПараметр;
    КонецЕсли;

    Если Данные = Неопределено Тогда
            ОтветHTTP = Соединение.Получить(ЗапросHTTP);
    Иначе
            ОтветHTTP = Соединение.ОтправитьДляОбработки(ЗапросHTTP);
    КонецЕсли;
    РезультатЗапроса = "";
    Если ОтветHTTP.КодСостояния = 200 Тогда
            РезультатЗапроса = ОтветHTTP.ПолучитьТелоКакСтроку();
    Иначе
            Сообщение = Новый СообщениеПользователю;
            Сообщение.Текст = ОтветHTTP.ПолучитьТелоКакСтроку();
            Сообщение.Сообщить();
    КонецЕсли;
    Возврат РезультатЗапроса;
КонецФункции
#КонецОбласти
#Область ОбработкаЗапроса
Процедура ОбработатьКоманды(chat_id,ПоследнееОбновление) Экспорт
    НаборЗаписейНастроек = РегистрыСведений.ТГ_Настройки.СоздатьМенеджерЗаписи();
    НаборЗаписейНастроек.НалаштуванняМодуля = "ТГ_TelegramBot";
    МенеджерЗаписи = РегистрыСведений.ТГ_update.СоздатьМенеджерЗаписи();
    МенеджерЗаписи.update_id = ПоследнееОбновление;
    МенеджерЗаписи.Прочитать();
    Если МенеджерЗаписи.text="/start" Тогда
            Если chat_id = idАдминБота Тогда
                textOfMessage = "Доброго дня, шановний адміністратор, " +
МенеджерЗаписи.first_name;
                SendMessage(chat_id,textOfMessage);
            Иначе
                textOfMessage = "Вас вітає бот для вивантаження документів та актів! " +
Символы.ПС +
```

```
                "Для отримання детальнішої інформації, відправте ваш номер телефона: ";

                SendMessage(chat_id,textOfMessage,СгенерироватьКлавиатуруНомер());
            КонецЕсли;
        ИначеЕсли МенеджерЗаписи.text = "/have_contact" Тогда
            // Команда выполяется в случае совпадение текста последнего обновления с текстом /have_contact
            НомерТелефонаСоответсвие = Ложь;
            КодЕДРПОУ = "";
            КонтрагентОбъект = Справочники.Контрагенты.Выбрать();
            НомерТелефонаЗапрос = "";
            Пока КонтрагентОбъект.Следующий() Цикл
                    КонецЕсли;
                КонецЕсли;
                МенеджерЗаписиТелеграмПользователи.Записать();
                textCounterParty =  "За вашим номером знайдено наступних контрагентів, оберіть одного для подальшої роботи: ";
        SendMessage(chat_id,textCounterParty,СгенеироватьИнлайнКлавиатуруСМассивом("counterparty_",МассивКонтрагентов));
            Иначе
                ОтправитьСообщениеАдминистратору(chat_id);
                text = "Номер телефона не знайдено ! " + Символы.ПС +
                "Адмінстратор зв"яжеться з вами найближчим часом." + Символы.ПС +
                "Після розмови з ним, наберіть команду /start.";
                SendMessage(chat_id,text);
            КонецЕсли;
        ИначеЕсли Найти(МенеджерЗаписи.text,"counterparty_") = Истина Тогда
            ДлинаПодстроки = СтрДлина("counterparty_");
            Контрагент = Сред(МенеджерЗаписи.text,ДлинаПодстроки + 1);
            КонтрагентСсылка = ЗапросСсылкаКонтрагентПоНаименованию(Контрагент);
            МассивОрганизаций = ЗапросПолучениеСвязанныхОрганизацийСКонтрагентом(КонтрагентСсылка);
            chosenCounterPartyText = "Обраний вами контрагент: " + Контрагент;
            SendMessage(chat_id,chosenCounterPartyText);  Если МассивОрганизаций.Количество() > 0 Тогда
                textOfOrganization  = "Знайдені зв'язки з наступними організаціями, оберіть одну з нижчевказаних: ";

        SendMessage(chat_id,textOfOrganization,СгенеироватьИнлайнКлавиатуруСМассивом("organization_",МассивОрганизаций));
        ИначеЕсли Найти(МенеджерЗаписи.text,"organization_") = Истина  Тогда
```

```
            ДлинаПодстроки = СтрДлина("organization_");
            Организация = Сред(МенеджерЗаписи.text,ДлинаПодстроки + 1);
            chosenOrganizationText = "Обрана вами організація: " +   Организация;
            ОрганизацияСсылка = ЗапросСсылкаОрганизацияПоНаименованию(Организация);
            ЗаписьЭлементов = РегистрыСведений.ТГ_ПараметрыБота.СоздатьМенеджерЗаписи();
            ЗаписьЭлементов.ВыбранныеЭлементы = "ТекущийВыбранныйЭлемент";
            ЗаписьЭлементов.Прочитать();
            Контрагент  = ЗаписьЭлементов.Контрагент;
            ЗаписьЭлементов.Организация = ОрганизацияСсылка;
            ЗаписьЭлементов.Контрагент = Контрагент;
            ЗаписьЭлементов.Записать();
            SendMessage(chat_id,chosenOrganizationText);

            textPeriod = "Оберіть необхідний період: ";
            МассивГод = Новый Массив;
            МассивГод.Добавить("За минулий рік");
            МассивГод.Добавить("За поточний рік");
    SendMessage(chat_id,textPeriod,СгенеироватьИнлайнКлавиатуруСМассивом("year_",МассивГод))
            ДлинаПодстроки = СтрДлина("year_");
            ТипПериода = Сред(МенеджерЗаписи.text,ДлинаПодстроки + 1);
            СоздатьАктСверки(chat_id,КонтрагентСсылка,ОрганизацияСсылка,ТипПериода);
    КонецЕсли;
КонецПроцедуры
#КонецОбласти
#Область МетодыЗапроса
Процедура GetUpdates(offset=0,limit=0,timeout=0) Экспорт
    chat_id=0;
    ПоследнееОбновление = "";
    // upd
    // изменена процедура,запрос формируется исходя из последнего значения регистра сведений
    // если значений в регистре нету, то offset=0
    // удален лишний запрос и таблица значений
    // код стал короче,понятнее и нету лишних вызовов
    ПараметрыМетода = Новый Массив;
    Если ЗапросПоследнегоОбновления() <> Null Тогда
            offset = Число(ЗапросПоследнегоОбновления()) + 1;
            ПоследнееОбновление = offset;
```

```
КонецЕсли;
ПараметрыМетода.Добавить("?offset=" + Формат(offset,"ЧРГ=' '; ЧГ=0"));
РезультатДляЗаписи = ОтправитьHTTPЗапрос("getUpdates",ПараметрыМетода,);
ЧтениеJSON = Новый ЧтениеJSON;
ЧтениеJSON.УстановитьСтроку(РезультатДляЗаписи);
ОтветСоотв = ПрочитатьJSON(ЧтениеJSON,Истина);
Для Каждого upd Из ОтветСоотв["result"] Цикл
        Если upd["message"] <> Неопределено Тогда
                message = upd["message"];
                мз = РегистрыСведений.ТГ_update.СоздатьМенеджерЗаписи();
                мз.update_id = upd["update_id"];
                мз.date = ТекущаяДата();
                мз.text = message["text"];
                мз.message_id     = message["message_id"];
                мз.first_name      = message["from"]["first_name"];
                мз.id                        = message["from"]["id"];
                мз.Записать();
                Контакт = message.Получить("contact");
                Если Контакт <> Неопределено Тогда
                        Если      ПользователяЕщеНетВЗарегистрированных(message["from"]["id"])
Тогда

    ИзменитьРегПользователиТелеграммПроходящиеРегистрацию(message["from"]["id"],Контакт["phon
e_number"]);
                                мз.text = "/have_contact";
                        Иначе
                                мз.text = "/have_contact";
                        КонецЕсли;
                КонецЕсли;
                мз.Записать();
                ОбработатьКоманды(message["from"]["id"],ПоследнееОбновление);
        КонецЕсли;
        Если upd["callback_query"] <> Неопределено Тогда
                Ответ = upd["callback_query"];
                мз = РегистрыСведений.ТГ_update.СоздатьМенеджерЗаписи();
                мз.update_id = upd["update_id"];
                мз.date = ТекущаяДата();
                мз.text = Ответ["data"];
```

```
                    мз.first_name      = Ответ["from"]["first_name"];
                    мз.id                        = Ответ["from"]["id"];
                    мз.Записать();
                    ОбработатьКоманды(Ответ["from"]["id"],ПоследнееОбновление);
                КонецЕсли;
        КонецЦикла;
КонецПроцедуры
Функция SendMessage(chat_id,text,reply_markup = Неопределено) Экспорт
        ПараметрыМетода = Новый Массив;
        ПараметрыМетода.Добавить("?chat_id=" + ФорматироватьСтроку(chat_id));
        ПараметрыМетода.Добавить("&text=" + СокрЛП(text));
        Если reply_markup <> Неопределено Тогда
                ПараметрыМетода.Добавить("&reply_markup=" + reply_markup);
        КонецЕсли;
        Результат = ОтправитьHTTPЗапрос("sendMessage",ПараметрыМетода,);
        Возврат Результат;
КонецФункции
Функция СгенеироватьИнлайнКлавиатуруСМассивом(Префикс,МассивТекстовКнопок)
        Клавиатура = ''
    |{
    |""inline_keyboard"": [";
    Для Каждого ТекстКнопка Из МассивТекстовКнопок Цикл
        Клавиатура = Клавиатура + ''
        | [
        |  {
        |    ""text"": """ + ТекстКнопка + """,
        |    ""callback_data"": """ + Префикс + "" + ТекстКнопка + """
        |  }
        | ],";
    КонецЦикла;
    Если Прав(Клавиатура, 1) = "," Тогда
        Клавиатура = Лев(Клавиатура, СтрДлина(Клавиатура) - 1);
    КонецЕсли;
    Клавиатура = Клавиатура + ''
    |]
    |}";
    Возврат Клавиатура;
```

```
КонецФункции

Функция СгенерироватьИнлайнКлавиатуруОдинЭлемент(Префикс, ТекстКнопка)
    Клавиатура = "{
        | ""inline_keyboard"": [
        |   [
        |     {
        |       ""text"": """ + ТекстКнопка + """,
        |       ""callback_data"": """ + Префикс +  """
        |     }
        |   ]
        | ]
        |}";
    Возврат Клавиатура;
КонецФункции

Процедура СоздатьАктСверки(chat_id,КонтрагентССылка,ОрганизацияССылка,ТипПериода)
      ГодНачало = "";
      ГодКонец = "";
      Если ТипПериода = "За минулий рік" Тогда
                  ПрошлыйГод = Год(ТекущаяДата()) - 1;
                  ГодНачало = Дата(ПрошлыйГод,01,01);
                  ГодКонец = Дата(ПрошлыйГод,12,31);
      ИначеЕсли ТипПериода = "За поточний рік" Тогда
                  ТекущийГод = Год(ТекущаяДата());
                  ГодНачало = Дата(ТекущийГод,01,01);
                  ГодКонец = Дата(ТекущийГод,12,31);
      КонецЕсли;
      АктСверки = Документы.АктСверкиВзаиморасчетов.СоздатьДокумент();
      АктСверки.Дата = ТекущаяДата();
      АктСверки.Контрагент = КонтрагентССылка;
      АктСверки.Организация = ОрганизацияССылка;
      ССылкаВалюта = Справочники.Валюты.НайтиПоКоду("980");
      АктСверки.ВалютаДокумента = ССылкаВалюта;
      МассивЗначенийСчетов = Новый Массив;
      МассивЗначенийСчетов.Добавить(Новый
Структура("Счет,УчаствуетВРасчетах",ПланыСчетов.Хозрасчетный.НайтиПоКоду("36"),Истина));
      МассивЗначенийСчетов.Добавить(Новый
Структура("Счет,УчаствуетВРасчетах",ПланыСчетов.Хозрасчетный.НайтиПоКоду("371"),Истина));
```

```
МассивЗначенийСчетов.Добавить(Новый
Структура("Счет,УчаствуетВРасчетах",ПланыСчетов.Хозрасчетный.НайтиПоКоду("377"),Истина));

МассивЗначенийСчетов.Добавить(Новый
Структура("Счет,УчаствуетВРасчетах",ПланыСчетов.Хозрасчетный.НайтиПоКоду("63"),Истина));

МассивЗначенийСчетов.Добавить(Новый
Структура("Счет,УчаствуетВРасчетах",ПланыСчетов.Хозрасчетный.НайтиПоКоду("681"),Истина));

МассивЗначенийСчетов.Добавить(Новый
Структура("Счет,УчаствуетВРасчетах",ПланыСчетов.Хозрасчетный.НайтиПоКоду("685"),Истина));

ФильтрСписокСчетов = Новый Массив;

Для Каждого Элемент из МассивЗначенийСчетов Цикл

        СписокСчетов = АктСверки.СписокСчетов.Добавить();

        СписокСчетов.Счет = Элемент.Счет;

        СписокСчетов.УчаствуетВРасчетах = Элемент.УчаствуетВРасчетах;

        Если Элемент.УчаствуетВРасчетах Тогда

                ФильтрСписокСчетов.Добавить(Элемент.Счет);

        КонецЕсли;

КонецЦикла;

СтруктураПараметров = Новый Структура;

АктСверки.ДатаНачала = ГодНачало;

АктСверки.ДатаОкончания = ГодКонец;

СтруктураПараметров.Вставить("ДатаНачала",ГодНачало);

СтруктураПараметров.Вставить("ДатаОкончания",Новый Граница(КонецДня(ГодКонец),
ВидГраницы.Включая));

АктСверки.ПоДаннымОрганизации.Очистить();

АктСверки.ПоДаннымКонтрагента.Очистить();

СтруктураПараметров.Вставить("Организация", ОрганизацияСсылка);

СтруктураПараметров.Вставить("Контрагент", КонтрагентСсылка);

СтруктураПараметров.Вставить("ЗаполнятьДанныеКонтрагента", Ложь);

СтруктураПараметров.Вставить("Валюта", Неопределено);

СтруктураПараметров.Вставить("ФильтрСписокСчетов", ФильтрСписокСчетов);

АналитикаРасчетов = Новый Массив;

АналитикаРасчетов.Добавить(ПланыВидовХарактеристик.ВидыСубконтоХозрасчетные.Контрагент
ы);

АналитикаРасчетов.Добавить(ПланыВидовХарактеристик.ВидыСубконтоХозрасчетные.Договоры);

СтруктураПараметров.Вставить("АналитикаРасчетов", АналитикаРасчетов);

СтруктураПараметров.Вставить("ВыводитьПолныеНазванияДокументов", Ложь);

СтруктураПараметров.Вставить("ВалютаДокумента", ССылкаВалюта);

СтруктураПараметров.Вставить("РазбитьПоДоговорам", Ложь);

СтруктураПараметров.Вставить("ВалютаРегламентированногоУчета", ССылкаВалюта);

СтруктураПараметров.Вставить("ДоговорКонтрагента", Неопределено);
```

```
    СтруктураДанных = ПодготовитьДанныеДляЗаполнения(СтруктураПараметров);
    Контрагент = ЗапросНаименованиеКонтрагентПоСсылке(КонтрагентСсылка);
    Организация = ЗапросНаименованиеОрганизацияПоСсылке(ОрганизацияСсылка);
    Если СтруктураДанных <> Неопределено Тогда
        АктСверки.ОстатокНаНачало = СтруктураДанных.ОстатокНаНачало;
        АктСверки.ПоДаннымОрганизации.Загрузить(СтруктураДанных.ПоДаннымОрганизации);
        АктСверки.Записать();
        Сообщить("Акт звірки за контрагентом " + Контрагент + " та організацією " + Организация
+ " створено.");
        СоответствиеАктаСверки =
ВывестиСписокАктовСверки(КонтрагентСсылка,ОрганизацияСсылка,ГодНачало,ГодКонец);

    SendDocument(chat_id,,СоответствиеАктаСверки["Заголовок"],СоответствиеАктаСверки["ИмяФайл
аПолное"]);
    Иначе
        textOfReconcialtion = "Не знайдено розрахунків між контрагентом " + Контрагент + " та
організацією " + Организация;
        ЭмодзиДом = "&#128513";

    SendMessage(chat_id,textOfReconcialtion,СгенерироватьИнлайнКлавиатуруОдинЭлемент("/start","На
початок"));
    КонецЕсли;
КонецПроцедуры

Функция ИзменитьРегПользователиТелеграммПроходящиеРегистрацию(id, НомерТелефона =
Неопределено)
    МенеджерЗаписи =
РегистрыСведений.ТГ_ПользователиТелеграммПроходящиеРегистрацию.СоздатьМенеджерЗаписи();
    МенеджерЗаписи.idПользователя=  id;
    МенеджерЗаписи.Прочитать();
    МенеджерЗаписи.idПользователя = id;
    Если НомерТелефона <> Неопределено Тогда
        МенеджерЗаписи.НомерТелефона = НомерТелефона;
    Иначе
        МенеджерЗаписи.НомерТелефона = "";
    КонецЕсли;
    МенеджерЗаписи.Записать();
КонецФункции
Функция ПользователяЕщеНетВЗарегистрированных(idПользователя)
    ПользовательЗарегистрирован = Истина;
    Запрос = Новый Запрос;
```

```
    Запрос.Текст =
        "ВЫБРАТЬ
        |    ТГ_ПользователиТелеграмм.idПользователя КАК idПользователя
        |ИЗ
        |    РегистрСведений.ТГ_ПользователиТелеграмм КАК ТГ_ПользователиТелеграмм
        |ГДЕ
        |    ТГ_ПользователиТелеграмм.idПользователя = &idПользователя";
    Запрос.УстановитьПараметр("idПользователя", idПользователя);
    РезультатЗапроса = Запрос.Выполнить();
    ВыборкаДетальныеЗаписи = РезультатЗапроса.Выбрать();
    Пока ВыборкаДетальныеЗаписи.Следующий() Цикл
        ПользовательЗарегистрирован= Ложь;
        Прервать;
    КонецЦикла;
    Возврат ПользовательЗарегистрирован;
КонецФункции
Функция ПодготовитьДанныеДляЗаполнения(СтруктураПараметров) Экспорт
        УстановитьПривилегированныйРежим(Истина);
    СтруктураДанныхЗаполнения = Новый Структура();
    СтруктураДанныхЗаполнения.Вставить("Успешно", Ложь);
    КодЯзыкаПечать = Локализация.ПолучитьЯзыкФормированияПечатныхФорм();
    //СчетаИсключения          = СчетаИсключения();
    ТаблицаПоДаннымОрганизации = НовыйТаблицаДокументов();
    КэшМетаданных              = Новый Соответствие; // Используется для получения представления документов
    Запрос = ПодготовитьЗапрос(СтруктураПараметров);
    РезультатЗапроса = Запрос.ВыполнитьПакет();
    ВыборкаОстатокНаНачало = РезультатЗапроса[1].Выбрать();
    ВыборкаОстатокНаНачало.Следующий();
    СтруктураДанныхЗаполнения.Вставить("ОстатокНаНачало",
ВыборкаОстатокНаНачало.ОстатокНаНачало);
    Выборка = РезультатЗапроса[2].Выбрать();
    Проверка = 0;
    Пока Выборка.Следующий() Цикл
        Проверка = 1;
        Если Выборка.Дебет = 0 И Выборка.Кредит = 0 Тогда
            Продолжить;
        КонецЕсли;
```

```
            НоваяСтрока = ТаблицаПоДаннымОрганизации.Добавить();
            ЗаполнитьЗначенияСвойств(НоваяСтрока, Выборка);
            ПроверитьДанныеСтроки(НоваяСтрока, Выборка);
            НоваяСтрока.Представление = ПредставлениеДокумента(
                  СтруктураПараметров,
                  Выборка,
                  КэшМетаданных,
                  КодЯзыкаПечать);
      КонецЦикла;
      Если Проверка = 0 Тогда
            Возврат Неопределено;
      КонецЕсли;
      ДополнитьТаблицуВалютнойСуммой(ТаблицаПоДаннымОрганизации, СтруктураПараметров);
      КолонкиДляСортировки = ?(СтруктураПараметров.РазбитьПоДоговорам,
            "Договор, Дата, Документ, Представление",
            "Дата, Документ, Представление, Договор");
      ТаблицаПоДаннымОрганизации.Сортировать(КолонкиДляСортировки, Новый СравнениеЗначений);
      СтруктураДанныхЗаполнения.Вставить("ПоДаннымОрганизации",
ТаблицаПоДаннымОрганизации);
      Возврат СтруктураДанныхЗаполнения;
КонецФункции
Функция НовыйТаблицаДокументов()
      ТаблицаДокументов = Новый ТаблицаЗначений;
      ТаблицаДокументов.Колонки.Добавить("Договор",                                    Новый
ОписаниеТипов("СправочникСсылка.ДоговорыКонтрагентов"));
      ТаблицаДокументов.Колонки.Добавить("Дата",
ОбщегоНазначения.ОписаниеТипаДата(ЧастиДаты.Дата));
      ТаблицаДокументов.Колонки.Добавить("Документ",      Документы.ТипВсеСсылки());
      ТаблицаДокументов.Колонки.Добавить("Представление",
ОбщегоНазначения.ОписаниеТипаСтрока(200));
      ТаблицаДокументов.Колонки.Добавить("Дебет",
ОбщегоНазначения.ОписаниеТипаЧисло(15,2));
      ТаблицаДокументов.Колонки.Добавить("Кредит",
ОбщегоНазначения.ОписаниеТипаЧисло(15,2));
      ТаблицаДокументов.Колонки.Добавить("Валюта",                                     Новый
ОписаниеТипов("СправочникСсылка.Валюты"));
      ТаблицаДокументов.Колонки.Добавить("ВалютнаяСумма",
ОбщегоНазначения.ОписаниеТипаЧисло(15,2));
      ТаблицаДокументов.Колонки.Добавить("ДатаВходящегоДокумента",
ОбщегоНазначения.ОписаниеТипаДата(ЧастиДаты.Дата));
      Возврат ТаблицаДокументов;
```

```
КонецФункции

Процедура ПроверитьДанныеСтроки(НоваяСтрока, Выборка)
    //проверят данные строки: может это на самом деле сторно НДС
    Если (ТипЗнч(Выборка.Документ) = Тип("ДокументСсылка.ВозвратТоваровПоставщику"))
            и (Выборка.КорСчет = ПланыСчетов.Хозрасчетный.НалоговыйКредитНеподтвержденный)
            и (Выборка.Кредит<0) Тогда
                НоваяСтрока.Дебет = -Выборка.Кредит;
                НоваяСтрока.Кредит =0;
    КонецЕсли;
КонецПроцедуры

Функция
ВывестиСписокАктовСверки(КонтрагентСсылка,ОрганизацияСсылка,ДатаНачала,ДатаОкончания)
    ДокументСсылка = "";

    //{{КОНСТРУКТОР_ЗАПРОСА_С_ОБРАБОТКОЙ_РЕЗУЛЬТАТА
    // Даний фрагмент побудований конструктором.
    // При повторному використанні конструктора, внесені вручну зміни будуть втрачені!!!

    Запрос = Новый Запрос;
    Запрос.Текст =
            "ВЫБРАТЬ
            |       АктСверкиВзаиморасчетов.Ссылка КАК Ссылка
            |ИЗ
            |       Документ.АктСверкиВзаиморасчетов КАК АктСверкиВзаиморасчетов
            |ГДЕ
            |       АктСверкиВзаиморасчетов.Контрагент.Ссылка = &КонтрагентСсылка
            |       И АктСверкиВзаиморасчетов.Организация.Ссылка = &ОрганизацияСсылка
            |       И АктСверкиВзаиморасчетов.ДатаНачала = &ДатаНачала
            |       И АктСверкиВзаиморасчетов.ДатаОкончания = &ДатаОкончания";

    Запрос.УстановитьПараметр("ДатаНачала", ДатаНачала);
    Запрос.УстановитьПараметр("ДатаОкончания", ДатаОкончания);
    Запрос.УстановитьПараметр("КонтрагентСсылка", КонтрагентСсылка);
    Запрос.УстановитьПараметр("ОрганизацияСсылка", ОрганизацияСсылка);

    РезультатЗапроса = Запрос.Выполнить();
```

```
    ВыборкаДетальныеЗаписи = РезультатЗапроса.Выбрать();

    Пока ВыборкаДетальныеЗаписи.Следующий() Цикл
            ДокументСсылка = ВыборкаДетальныеЗаписи.Ссылка;
    КонецЦикла;
    массивСсылок = Новый Массив;
    массивСсылок.Добавить(ДокументСсылка);

    //}}КОНСТРУКТОР_ЗАПРОСА_С_ОБРАБОТКОЙ_РЕЗУЛЬТАТА

    мОбъектыПечати = Новый Массив;
    мОбъектыПечати.Добавить(Документы.АктСверкиВзаиморасчетов);


     Результат = УправлениеПечатью.СформироватьПечатныеФормы("Документ.АктСверкиВзаиморасчетов",
"АктСверки",массивСсылок,мОбъектыПечати,Неопределено);
     Док = Результат.КоллекцияПечатныхФорм[0].ТабличныйДокумент;

      // Определяем путь для сохранения PDF-файла
      ПутьКФайлу = КаталогВременныхФайлов() + "" + "ReconcilationAct:" + Строка(Формат(Результат.ОбъектыПечати.Получить(0).Значение.Дата,"ДФ=dd.MM.yy")) + ".pdf" ;
      // Записываем табличный документ в формате PDF
      Док.Записать(ПутьКФайлу, ТипФайлаТабличногоДокумента.PDF);
      Сообщить("Файл успішно збережений: " + ПутьКФайлу);
      СоотвЗнач = Новый Соответствие;
      СоотвЗнач.Вставить("Заголовок","Акт звірки за " + Строка(Формат(Результат.ОбъектыПечати.Получить(0).Значение.Дата,"ДФ=dd.MM.yy")));
      СоотвЗнач.Вставить("ИмяФайлаПолное",ПутьКФайлу);
      Возврат СоотвЗнач;
КонецФункции

Функция ФорматироватьСтроку(ТекущееЗначение) Экспорт
    Возврат Формат(ТекущееЗначение, "ЧРГ="; ЧГ=0");
КонецФункции

Функция СформироватьТекстЗапросаДляДокумента(Данные)

    ТекстЗапроса = "";
```

```
    ТекстЗапроса = ТекстЗапроса + "--" + Данные["Boundary"] + Символы.ВК + Символы.ПС;

    ТекстЗапроса = ТекстЗапроса + "Content-Disposition: form-data; name=""" + Данные["name"] + """; filename=""" + Данные["ИмяФайла"] + """" + Символы.ВК + Символы.ПС;

    ТекстЗапроса = ТекстЗапроса + "Content-Type: application/x-zip-compressed" + Символы.ВК + Символы.ПС + Символы.ВК + Символы.ПС;


    ДвоичныеДанныеСтрокой = ПолучитьДвоичныеДанныеВСтрокуБезКодирования(Данные["ИмяФайлаПолное"]);


    ТекстЗапроса = ТекстЗапроса + ДвоичныеДанныеСтрокой + Символы.ВК + Символы.ПС;


    ТекстЗапроса = ТекстЗапроса + "--" + Данные["Boundary"] + "--" + Символы.ВК + Символы.ПС;


    Возврат ТекстЗапроса;
КонецФункции


Функция ПолучитьДвоичныеДанныеВСтрокуБезКодирования(ИмяФайлаПолное)
    ТекстовыйДокумент = Новый ТекстовыйДокумент;
    ТекстовыйДокумент.Прочитать(ИмяФайлаПолное, КодировкаТекста.ANSI, Символы.ПС);
    Возврат ТекстовыйДокумент.ПолучитьТекст();
КонецФункции



#КонецОбласти


#Область Запросы
```