

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПІДТРИМКИ

(назва теми)

ПЛАНІВ ПОДОРОЖЕЙ

TRIP PLANNING SUPPORT SOFTWARE

Виконав(ла): студент(ка) 4 курсу, групи КНТ-220сп

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

МИХАЙЛИЧЕНКО В.О.

(ПРИЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРИЗВИЩЕ та ініціали)

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
« » 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

МИХАЙЛИЧЕНКА Владислава Олександровича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення підтримки планів подорожей
Trip Planning Support Software

керівник проєкту (роботи) к.т.н., доцент СКРУПСЬКИЙ Степан Юрійович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «04» квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 1 червня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми.
3. Реалізація програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СКРУПСЬКИЙ С.Ю., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання « 4 » квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Срок виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Розробка архітектури програми	3 тиждень	Розділ 2
4	Розробка програми	4-5 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
7	Захист роботи	8 тиждень	

Студент(ка)

_____ Владислав МИХАЙЛИЧЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Степан СКРУПСЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 93 с., 24 рис., 2 табл., 3 дод., 11 джерел.

ПОДОРОЖ, ПЛАН ПОДОРОЖІ, ВИТРАТИ, ЕКСПЕРТ, РЕКОМЕНДАЦІЇ, ВЕБСИСТЕМА.

Об'єктом роботи є процес підтримки планування подорожей. Предметом роботи є програмне забезпечення підтримки планів подорожей. Мета роботи – розробити програмне забезпечення підтримки планів подорожей для забезпечення мандрівників відповідними засобами поза межами туристичних компаній.

Матеріали, методи та технічні засоби: мова Python, фреймворк Django, який підтримує веброзробку, система керування базами даних MySQL.

Результати. Вебсистема була розроблена шляхом аналізу аналогів зокрема і предметної області загалом, визначення сценаріїв роботи мандрівника й експерта, створення бази даних, реалізації засобами фреймворку Django.

Висновки. Мандрівники, які подорожують, не використовуючи послуги туристичних компаній, все одно потребують створення плану подорожі, придбання квитків, бронювання житла. Вони можуть створити цей план або самотійно, або використовуючи послуги експертів. Програмне забезпечення дозволяє знаходити експертів для планування подорожей, які можуть створювати план або плани подорожі, готувати всі необхідні бронювання для мандрівника. Також мандрівники можуть знаходити нові місця, додавати їх до структури подорожі, визначаючи таким чином вимоги.

Галузь використання – подорожі як індивідуальні, так і в складі групи переважно без підтримки туристичних компаній.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 93 pages, 24 figures, 2 tables, 3 appendixes, 11 sources.

JOURNEY, ITINERARY, COSTS, EXPERT, RECOMMENDATIONS, WEB SYSTEM.

The object of the work is the process of supporting trip planning. The subject of the work is trip planning support software. The goal of the work is to develop trip planning support software to provide travelers with the appropriate tools outside of travel companies.

Materials, methods and technical means: Python language, Django framework which supports web development, MySQL database management system.

The results. The web system was developed by analyzing analogues in particular and the subject area in general, defining work scenarios for a traveler and an expert, creating a database, and implementing software using the Django framework.

Conclusions. Travelers who travel without using the services of travel companies still need to create a travel plan, purchase tickets, and book accommodations. They can create a trip plan either by themselves or by using the services of experts. The software allows users to find travel planning experts who can create a journey plan or plans, prepare all necessary reservations for the traveler. Also, travelers can find new places, add them to the journey structure, thus defining the requirements.

The field of use is travel both individually and as part of a group, mainly without the support of travel companies.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області.....	9
1.1 Аналіз програмних засобів підтримки планів подорожей.....	9
1.2 Визначення концепції розроблюваної програми	12
1.3 Висновки за розділом 1 та постановка завдань роботи	14
2 Розробка архітектури програми.....	15
2.1 Вибір технологій розробки	15
2.2 Моделювання вимог до програми	16
2.3 Проєктування бази даних	20
2.4 Висновки за розділом 2	35
3 Реалізація програми	36
3.1 Структура класів	36
3.2 Програмна реалізація класів вебсистеми.....	37
3.3 Висновки за розділом 3	43
4 Експлуатація та тестування програми	44
4.1 Призначення програми	44
4.2 Умови виконання програми	44
4.3 Виконання і тестування програми.....	44
4.4 Висновки за розділом 4	51
Висновки	52
Перелік джерел посилання	53
Додаток А Технічне завдання	54
Додаток Б Текст програми	59
Додаток В Слайди презентації.....	87

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

MVT – Model-View-Template;

БД – база даних.

ВСТУП

Персоналізація подорожей є важливим фактором сучасного туризму: пропонуючи гнучкі умови, які можна пристосувати до потреб мандрівника, можна краще задовольнити клієнтів, це може призвести в підсумку до повторних звернень мандрівників, тому пріоритетом, як пріоритетом будь-якої сучасної бізнес-діяльності, має бути надання послуг, орієнтованих на клієнта [1]. Якщо план подорожі складений індивідуально, враховує інтереси мандрівника – це крок до вищої задоволеності мандрівника. Створити такий тур туристичній компанії складно, адже фактично всі такі тури є певними шаблонами. Ці шаблони можуть трохи відрізнятися, але вони не дозволяють абсолютно змінювати всю подорож під конкретного мандрівника.

Планування подорожей – це цілий процес, від якісного виконання якого залежить те, на скільки задоволеним в підсумку буде мандрівник. Для цього потрібно мати певні знання, досвід, вміти правильно об'єднувати різні складові у цілісний план. Будь-які помилки можуть призвести до того, що враження про всю подорож зіпсується, а також до проблем. Будь-які проблеми в новій країні можуть призводити до погіршення психологічного стану. Відповідно можна говорити про те, що це окремий і достатньо тривалий процес, особливо за відсутності знань і досвіду. У результаті мандрівник може мати бажання, але не мати часу все спланувати якісно. У такому випадку люди з досвідом можуть допомогти сформувати план, виконати потрібні бронювання, щоб мандрівник сконцентрувався на подорожі, на відпочинку. Саме на підтримку взаємодії між мандрівника і експертами, тобто людьми, здатними сформувати план подорожі, направлена розробка, яку заплановано в роботі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз програмних засобів підтримки планів подорожей

Туризм є одним з рушіїв економіки багатьох країн світу. Для деяких країн він є основою економіки, забезпечуючи велику частку. Тому зрозуміло, що програмних засобів, які спрямовані на дану галузь, є достатньо багато. Не всі вони стосуються планування подорожей, але і такі засоби присутні.

Вебсайт Triptile [2] (рис. 1.1) дозволяє створювати власні плани подорожей. Для цього надається можливість знайти місця, які можуть бути цікавими мандрівникам, а потім додавати їх до маршруту подорожі. До подорожі додаються варіанти переміщення між складовими, тобто польоти, переїзди. На кожен день можна обрати одну з готових програм. Тобто щодо розглянутої у вступі персоналізації є неузгодження.

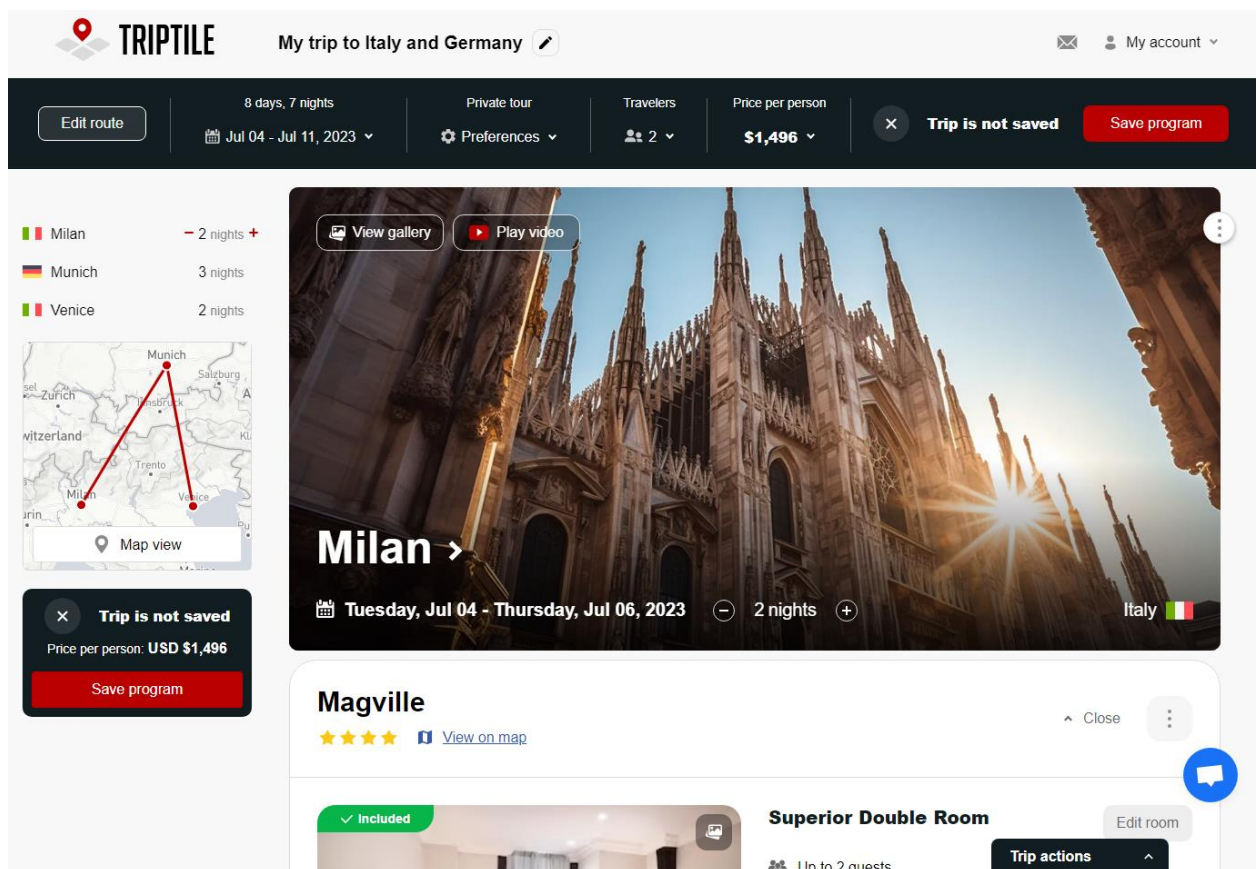


Рисунок 1.1 – Вебсайт Triptile

Надається можливість шукати готелі, що є основою даних про проживання. Однак проживання вносяться з обмеженого переліку готелів. Тобто фактично спланувати подорож, визначивши проживання в квартирі або подібне рішення, неможливо.

Вебсайт Wanderlog [3] (рис. 1.2) дозволяє вносити запланований маршрут подорожі, місця, які мандрівник бажає відвідати, місця проживання, переміщення (перельоти, переїзди). За кожним переліченим пунктом можна додавати витрати, а потім можна додавати ще інші витрати, розбиті за статтями, таким чином створюючи бюджет подорожі.

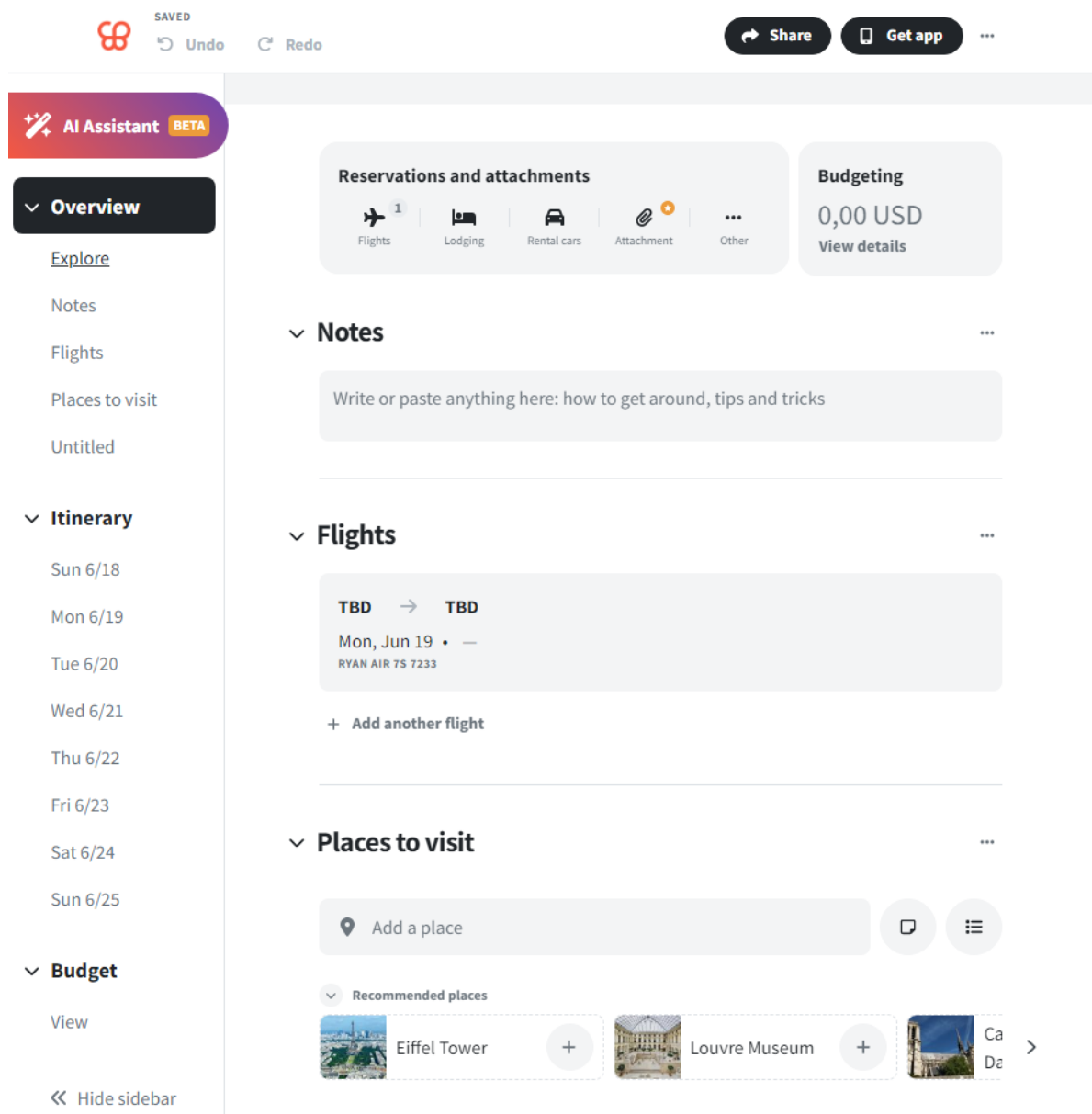


Рисунок 1.2 – Вебсайт Wanderlog

Щодо створення плану подорожі це достатньо корисний інструмент з великою кількістю можливостей. Проте він не надає засобів роботи в групі повноцінних, обговорення планів подорожей, а також не дозволяє комунікувати з людьми, які б створили для мандрівника план подорожі. Тобто мандрівнику тоді потрібно самому знайти десь експерта, а вже далі експерт зможе скористатися цим вебсайтом, щоб створити план подорожі.

Існує цілий ряд невеликих організацій, які надають можливість створити під мандрівника план подорожі. До них зокрема належать вебсайти [4]-[5], але перелік можна розширювати і далі. Проте все, що надають самі сайти, обмежується поданням заявки на створення плану, фактично надається загальна інформація і контактна форма.

Вебсайт Goasklocal [6] (рис. 1.3) дозволяє знайти консультанта, який проживає в країні, куди направляється мандрівник, і отримати у нього консультацію.

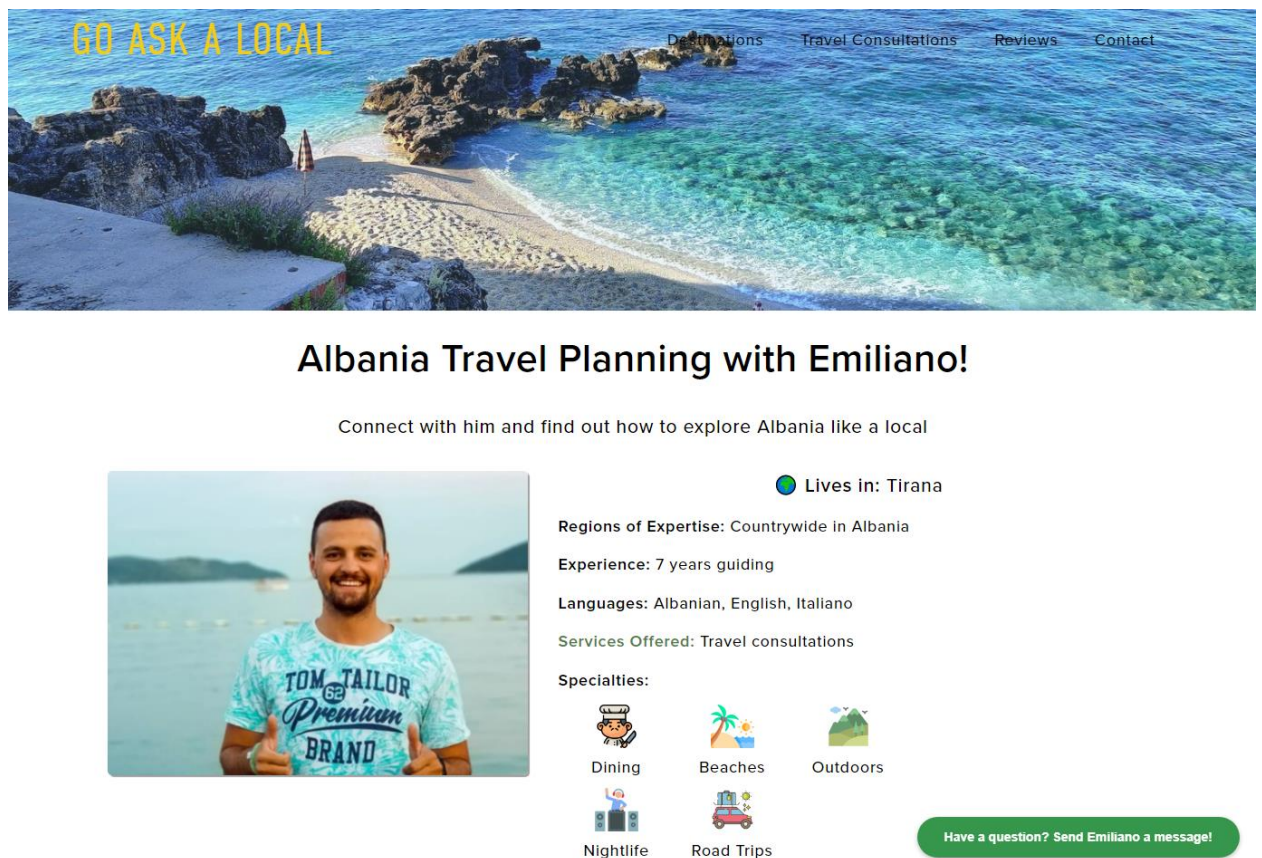


Рисунок 1.3 – Вебсайт Goasklocal

Сама консультація може відбуватися в zoom, наприклад, а вебсайт регламентує сплату за послуги, що виконується на погодинній основі. Тож можна знайти експерта, але подальша робота з ним відбувається поза межами вебсайту, до того ж робота з експертом включає запитання до нього і його відповіді, тобто далеко не обов'язково, що експерт погодиться створити план подорожі, це швидше виняток.

Вебсайт Global Game Plan [7] (рис. 1.4) дозволяє фактично створити запит на створення плану подорожі, визначаючи в формі особливості цієї подорожі. Форма є достатньо розлогою, проте в підсумку сам вебсайт надає можливість тільки сформулювати заявку.

Step 5: Have a great vacation!

Get Started With Travel Planning

Name (required)
First Name Last Name

Email (required)

Preferred contact method (required)
☐ Email
☐ Phone

Dates of your trip (required)

Who is going on the trip? (required)

Destination(s) if known (required)

Type of trip (required)
Check all that apply
☐ Domestic travel
☐ International travel
☐ Road trip
☐ Multi-family trip
☐ Special occasion trip (birthday, anniversary, etc.)
☐ Virtual Trip (heritage tour, date night, Zoom party)
☐ Other (please explain in comments)

Рисунок 1.4 – Вебсайт Global Game Plan

1.2 Визначення концепції розроблюваної програми

Враховуючи описані характеристики наявних аналогів, основою концепції розробки має стати можливість формування плану подорожі зі

всіма необхідними для цього функціями, пошуку місць, щоб їх зручно було знаходити і додавати до подорожі, але також і доступ до експертів, які б могли за необхідності такі плани створювати, персоналізуючи їх під вимоги конкретного мандрівника (табл. 1.1).

Аналіз показав, що на даний момент частина аналогів орієнтована на створення планів подорожей, а частина – на пошук експертів, які б могли надати інформацію про подорожі або створити плани подорожей. У другому випадку програми не надають можливостей створення в них планів. Тому важливим є об'єднання цих засобів, що буде сприяти зручності роботи, а через це допомогти взаємодії між мандрівниками і експертами.

Таблиця 1.1 – Результати аналізу програмних засобів підтримки планів подорожей

Ознака	Triptile	Wanderlog	Goasklocal	Global Game Plan	Розробка
Створення плану подорожі	Так	Так	Ні	Ні	Так
Формування бюджету подорожі	Так	Так	Ні	Ні	Так
Пошук експертів	Ні	Ні	Так	Так	Так
Реєстрація експертів	Ні	Ні	Ні	Ні	Так
Розділення роботи з подорожжю з мандрівниками	Ні	Ні	Ні	Ні	Так

1.3 Висновки за розділом 1 та постановка завдань роботи

Розробка має об'єднати можливості створення планів подорожей з можливостями розділення доступу до цих планів, щоб ними можна було певним чином керувати разом групі мандрівників, та з можливостями доступу до ринку послуг експертів, які надають послуги планування, щоб вони, а не сам мандрівник, могли створити план подорожі, виконати потрібні бронювання для цього, а мандрівник міг тільки відпочивати або подорожувати за цим планом за подібною логікою до туристичних компаній, але зі значно більшою персоналізацією подорожі.

Завдання роботи:

- аналіз програмних засобів підтримки планів подорожей і визначення концепції програми;
- проєктування програмного забезпечення підтримки планів подорожей;
- реалізація вебдодатку підтримки планів подорожей.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір технологій розробки

Як PHP, так і Python [8] є відомими на практиці мовами розробки вебсистем, хоча кожна з цих мов має свої переваги і недоліки (табл. 2.1).

При розробці даної програми важливим є спрощення процесу розробки, адже за результатами аналізу можна стверджувати, що потрібна достатньо велика кількість функцій у роботі програми, а час розробки обмежений строками, зазначеними в завданні на роботу, а також єдиним розробником. Враховуючи це, а також краще керування вразливостями зокрема і за рахунок засобів, вбудованих у вебфреймворк Django, перспективи розширення за рахунок моделей штучного інтелекту, що може знадобитися для спрощення планування в подальшому, розробка буде виконуватися на Python з фреймворком Django [9].

Таблиця 2.1 – Порівняння мов Python і PHP для розробки програми

Ознака	Python	PHP
Забезпечення потужного вебфреймворку	Так	Так
Спрощення процесу розробки	Вище	Менше
Керування вразливостями	Краще	Гірше
Можливості розширення	Перспективніші за рахунок простішої роботи з засобами штучного інтелекту	Менш перспективні в даному розрізі

2.2 Моделювання вимог до програми

Першою роллю користувача в програмі є мандрівник, що може скористатися такими функціями:

- пошук експертів;
- представлення рейтингу експертів;
- рекомендування місця;
- створення нової подорожі;
- редагування даних існуючої подорожі;
- представлення авторських подорожей;
- представлення доступних подорожей;
- надання доступу до подорожі;
- керування доступом до подорожі;
- відправлення запиту експерту за подорожжю;
- представлення запитів за подорожжю;
- представлення планів за подорожжю;
- голосування за план подорожі;
- обговорення планів подорожей;
- створення відгуку про взаємодію з експертом;
- пошук місць;
- представлення рейтингу місць у місті;
- представлення рейтингу місць загалом;
- представлення рейтингу місць за типом;
- створення нового плану подорожі;
- редагування існуючого плану подорожі;
- визначення маршруту за планом подорожі;
- реєстрація та авторизація;
- додавання статей витрат до плану подорожі;
- редагування статей витрат у плані подорожі;
- представлення бюджету за планом подорожі;

- представлення маршруту за планом подорожі;
- представлення проживань за планом подорожі;
- скасування запиту;
- узгодження статті плану подорожі;
- скасування статті плану подорожі;
- оплата послуг експерта;
- додавання нового бронювання проживання до плану подорожі;
- додавання нового бронювання прокату автомобіля до плану подорожі;
- додавання нового бронювання переміщення за маршрутом подорожі до плану;
- редагування існуючого бронювання проживання;
- редагування існуючого бронювання прокату автомобіля до плану подорожі;
- редагування існуючого бронювання переміщення;
- представлення рейтингу міст;
- представлення інформації про місце;
- представлення інформації про експерта;
- представлення інформації про мандрівника;
- представлення інформації про подорож;
- представлення інформації про план подорожі;
- додавання місця до подорожі;
- вилучення місця з подорожі.

Експерт має частину спільних функцій з мандрівником, але має і власні засоби. Експерт може скористатися такими функціями:

- пошук місць;
- представлення рейтингу міст;
- представлення рейтингу експертів;
- представлення планів за подорожжю;
- обговорення планів подорожей;

- створення відгуку про взаємодію з мандрівником;
- визначення цін на послуги;
- надання знижки на послуги;
- представлення рейтингу місць у місті;
- представлення рейтингу місць загалом;
- представлення рейтингу місць за типом;
- створення нового плану подорожі;
- редагування існуючого плану подорожі;
- визначення маршруту за планом подорожі;
- реєстрація та авторизація;
- додавання статей витрат до плану подорожі;
- редагування статей витрат у плані подорожі;
- представлення бюджету за планом подорожі;
- представлення маршруту за планом подорожі;
- представлення проживань за планом подорожі;
- представлення запитів до експерта;
- узгодження запиту;
- скасування запиту;
- скасування статті плану подорожі;
- представлення інформації про план подорожі;
- додавання нового бронювання проживання до плану подорожі;
- додавання нового бронювання прокату автомобіля до плану подорожі;
- додавання нового бронювання переміщення за маршрутом подорожі до плану;
- редагування існуючого бронювання проживання;
- редагування існуючого бронювання прокату автомобіля до плану подорожі;
- редагування існуючого бронювання переміщення;
- представлення інформації про місце (рис. 2.1).

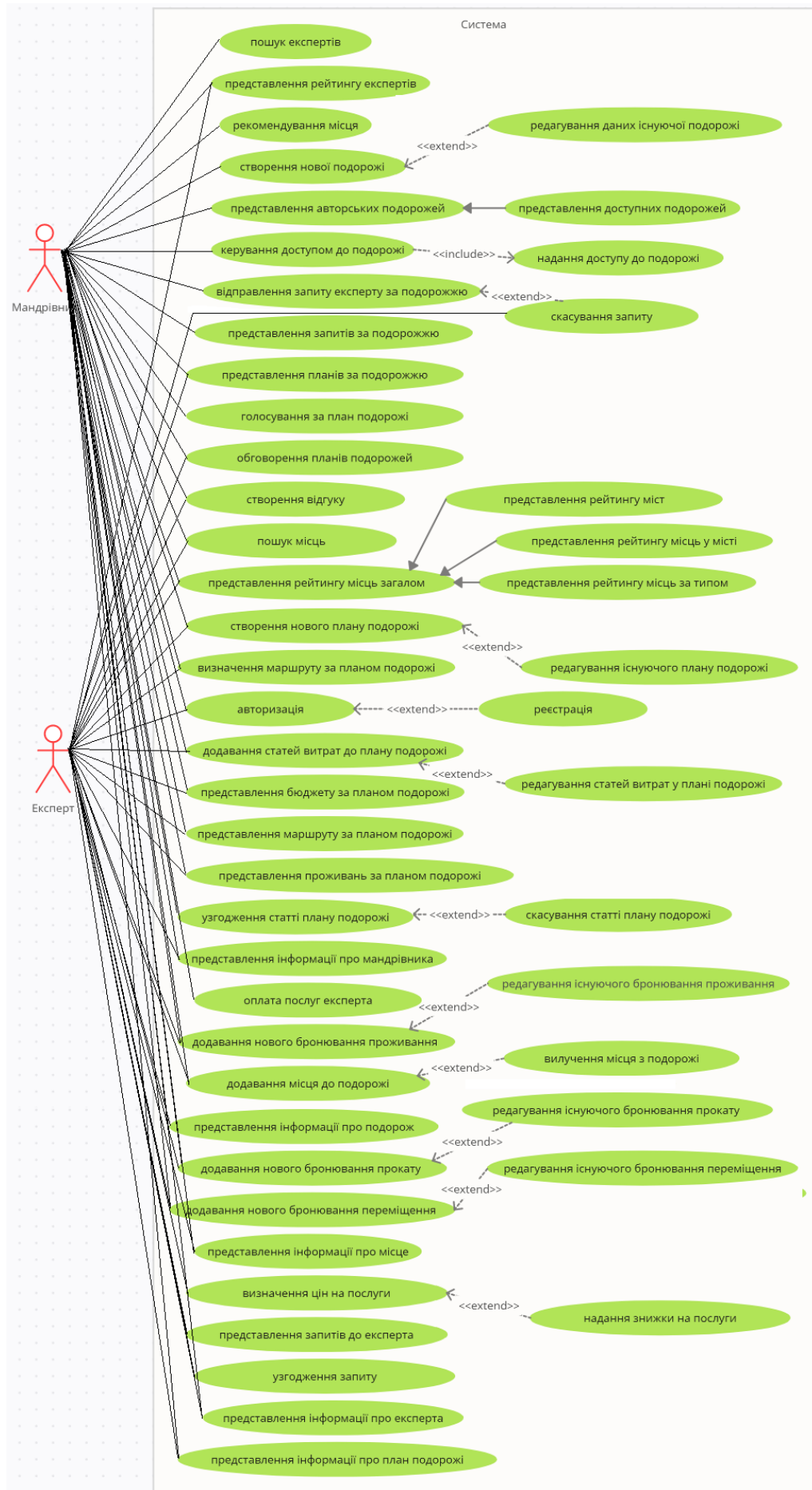


Рисунок 2.1 – Діаграма прецедентів

Також до функцій експерта належать:

- представлення інформації про експерта;
- представлення інформації про мандрівника;
- представлення інформації про подорож;
- додавання місця до подорожі;
- вилучення місць з подорожі.

Для проєктування інтерфейсів програми за цими сценаріями використано шаблон [10].

2.3 Проєктування бази даних

Розробка бази даних (БД) виконана засоби системи MySQL для роботи з БД.

Користувацькі облікові записи необхідні для підтримки роботи користувачів з програмою, тому їх зведено в таблицю БД, що має назву User і поля:

- id для ідентифікації за номером облікового запису;
- username для логіну користувач;
- password для паролю користувача;
- first_name для імені власника облікового запису;
- last_name для прізвища власника облікового запису.

Маючи облікові записи в програмі, усі користувачі в подальшому розбиваються на 2 групи: експертів, які виконуються планування подорожей, та мандрівників, які потребують допомоги з плануванням подорожей.

Дані експертів зведено в таблицю, що має назву Planner і поля:

- id для ідентифікації експерта;
- user для визначення облікового запису, яким користується цей експерт: зовнішній ключ, пов'язує дану таблицю з таблицею User;
- presentation для опису власних знань, вмінь експертом;
- personalphoto для файлу фотографії експерта;

- specialization для визначення загальної спеціалізації експерта щодо планування подорожей;

- mark для оцінки експерта від мандрівників, які скористалися послугами експерта для планування власних подорожей.

Дані мандрівників зведено в таблицю, що має назву Traveler і поля:

- id для ідентифікації мандрівника;
- user для визначення облікового запису, яким користується цей мандрівник: зовнішній ключ, пов'язує дану таблицю з таблицею User;
- presentation для опису себе мандрівником;
- personalphoto для файлу фотографії мандрівника;
- mark для оцінки мандрівника від експертів, які взаємодіяли з ним, тобто у яких він замовляв планування власних подорожей.

Особливості ціноутворення щодо послуг експерта зведені в таблицю, що має назву Plannerservice і поля:

- id для ідентифікації особливості ціноутворення;
- planner для визначення експерта, якого стосується ця особливість: зовнішній ключ, пов'язує дану таблицю з таблицею Planner;
- details для детального опису особливості, тобто за яких умов використовується саме ця вартість;
- pay для вартості виконання послуг з планування за цих особливостей подорожі.

Можливі тимчасові знижки на послуги з планування подорожей зведені в таблицю БД, що має назву Servicepromotion і поля:

- id для ідентифікації знижки;
- plannerservice для визначення особливості ціноутворення, якої стосується ця знижка: зовнішній ключ, пов'язує дану таблицю з таблицею Plannerservice;
- pay для визначення вартості за тимчасовою знижкою;
- currentstate для визначення поточного стану цієї знижки, тобто вона діє або вже ні.

Безпосередньо подорожі, для яких буде виконуватися планування вносяться в БД окремо мандрівником і з цього внесення розпочинається серія подальших процедур, які в результаті призводять до виникнення готового плану. Подорожі мандрівників зведені в таблицю, що має назву Journey і поля:

- id для ідентифікації подорожі;
- createdby для визначення мандрівника, який створив запису про цю подорож: зовнішній ключ, пов'язує дану таблицю з таблицею Traveler;
- createdat для збереження дати створення подорожі;
- startdate для збереження дати початку подорожі;
- enddate для збереження дати закінчення подорожі;
- details для збереження детального опису подорожі;
- maintitle для збереження назви подорожі.

Усі місця, через які може проходити подорож, яких вона може стосуватися, винесені разом. Тобто це можуть бути і населені пункти, які входять до маршруту, і готелі, і ресторани. Всі вони зведені в таблицю, що має назву Point і поля:

- id для ідентифікації місця;
- caption для назви цього місця;
- details для детального опису місця;
- pointtype для визначення типу місця: зовнішній ключ, пов'язує дану таблицю з таблицею Pointtype;
- recommendations для кількості рекомендацій, висловлених цьому місцю;
- disrecommendations для кількості негативних рекомендацій, висловлених цьому місцю;
- site для визначення адреси вебсайту цього місця.

Місця можуть мати окремі адреси, але не всі. Якщо це ресторан або готель, то за ними потрібно зберігати адресу, але якщо це місто, то його

адреса визначається його назвою. Тому адреси місць винесені окремо і зведені в таблицю, що має назву Pointaddress і поля:

- id для ідентифікації адреси місця;
- point для визначення місця, адреса якого визначена: зовнішній ключ, пов'язує дану таблицю з таблицею Point;
- citypoint для визначення населеного пункту, якому відповідає ця адреса: зовнішній ключ, пов'язує дану таблицю з таблицею Point;
- address для текстового представлення адреси;
- addressx для широти координати адреси;
- addressy для довготи координати адреси.

Усі місця можуть мати фотографії, для того щоб з ними ознайомитись, прийняти рішення, чи варто витратити час на це місце. Фотографій може бути багато, тому вони винесені в окрему таблицю, окремо від самого місця. Фотографії місць зведені в таблицю, що має назву Pointphoto і поля:

- id для ідентифікації фотографії місця;
- point для визначення місця, якому відповідає ця фотографія: зовнішній ключ, пов'язує дану таблицю з таблицею Point;
- photo для файлу фотографії місця.

Типи місць, якими як раз і визначається, що це населений пункт, ресторан, готель, краєвид або щось інше, зведені в таблицю, що має назву Pointtype і поля:

- id для ідентифікації типу місця;
- caption для назви місця.

Місця в програмі необхідні зокрема і для того, щоб їх додавати до подорожі. Внесення місць у подорож відбувається і самими мандрівниками, коли вони визначають, куди вони хочуть поїхати, звідки, а можливо і які саме місця вони хочуть відвідати у подорожі, а також і експертами, які додають місця до плану подорожей самостійно, якщо це потрібно. Місця подорожі зведені в таблицю, що має назву JourneyPoint і поля:

- id для ідентифікації присутності місця в подорожі;

- point для визначення місця, яке входить до подорожі: зовнішній ключ, пов’язує дану таблицю з таблицею Point;
- journey для визначення подорожі, до якої входить це місце: зовнішній ключ, пов’язує дану таблицю з таблицею Journey;
- byplanner для визначення того, хто вніс це місце до подорожі: мандрівник як вимогу або експерт самостійно;
- agreed для підтвердження мандрівником відвідування цього місця в подорожі.

На рівні плану подорожі кожне місце у подорожі конкретизується, якщо воно входить до певного плану, тобто визначається, в які строки відбудуться відвідини цього місця. Ці дані зводяться до таблиці БД, що має назву JourneyplanPoint і поля:

- id для ідентифікації місця в плані подорожі;
- journeypoint для ідентифікації входження місця в подорож, якого стосується ця конкретизація: зовнішній ключ, пов’язує дану таблицю з таблицею JourneyPoint;
- journeyplan для ідентифікації плану подорожі, до якого стосується ця конкретизація: зовнішній ключ, пов’язує дану таблицю з таблицею JourneyPlan;
- startdate для дати і/або часу, з якого плануються відвідини цього місця;
- enddate для дати і/або часу, до якого плануються відвідини цього місця;
- agreed для підтвердження мандрівником відвідування цього місця в подорожі саме в такі строки.

Ділянки маршруту подорожі зведені в таблицю, що має назву RouteSection і поля:

- id для ідентифікації ділянки маршруту подорожі;

- `startjourneyplanpoint` для визначення місця з плану, з якого починається ця ділянка: зовнішній ключ, пов'язує дану таблицю з таблицею `JourneyplanPoint`;

- `endjourneyplanpoint` для визначення місця з плану, в якому завершується ця ділянка: зовнішній ключ, пов'язує дану таблицю з таблицею `JourneyplanPoint`;

- `details` для детального опису стосовно цієї ділянки маршруту.

Витрати на подорож у плані зведені в таблицю, що має назву `Expense` і поля:

- `id` для ідентифікації витрати в плані подорожі;
- `journeyplan` для визначення плану подорожі, до якого відноситься ця витрата: зовнішній ключ, пов'язує дану таблицю з таблицею `JourneyPlan`;
- `caption` для загальної назви витрати;
- `details` для деталізації витрати;
- `level` для рівня витрати: вона є сумарною для цієї групи або вона є деталізацією сумарної (наприклад, транспортні витрати загалом або витрати на конкретні переміщення в межах одного міста);
- `expensegroup` для визначення групи витрат, до якої відноситься дана: зовнішній ключ, пов'язує дану таблицю з таблицею `Expensegroup`;
- `expansessum` для запланованої суми витрат;
- `payedsum` для визначення фактично сплаченої суми на даний момент: наприклад, квитки придбані наперед до подорожі.

Групи витрат (або іншими словами категорії) зведені в таблицю, що має назву `Expensegroup` і поля:

- `id` для ідентифікації групи витрат;
- `caption` для назви групи витрат.

Запити від мандрівників до експертів щодо створення плану подорожі зведені в таблицю, що має назву `PlannerRequest` і поля:

- `id` для ідентифікації запиту щодо створення плану подорожі;

- journey для визначення подорожі, план для якої потрібно створити: зовнішній ключ, пов'язує дану таблицю з таблицею Journey;
- planner для визначення експерта, запит до якого оформлено: зовнішній ключ, пов'язує дану таблицю з таблицею Planner;
- pointextended для вказання на можливість або неможливість внесення додаткових населених пунктів у план подорожі окрім тих, які вніс сам мандрівник;
- plannedonly для вказання на необхідність оформлення всіх попередніх дій або тільки планування, тобто наприклад, потрібно виконувати покупку квитків на літак чи ні, якщо вони передбачаються в плані подорожі;
- maxexpenses для визначення максимальних витрат на цю подорож, які готовий понести мандрівник;
- serviceprice для визначення вартості планування, визначеної експертом за його послуги;
- serviceday для визначення суми, яка за послуги експерта вже була сплачена на даний момент;
- createdat для визначення дати створення запиту;
- payedat для визначення дати сплати за послуги експерта;
- submitted для визначення прийняття експертом запиту на планування або скасування такого запиту, або відсутність відповіді;
- agreed для визначення затвердження плану подорожі мандрівником, який створив подорож, відсутності відповіді або скасування плану;
- requirements для вимог щодо плану, який необхідно створити.

Самі плани подорожі зведені в таблицю, що має назву JourneyPlan і поля:

- id для ідентифікації плану подорожі;
- plannerrequest для визначення запиту щодо створення плану подорожі, за яким було створено цей план: зовнішній ключ, пов'язує дану таблицю з таблицею PlannerRequest;
- caption для назви плану подорожі;

- details для детального опису особливостей плану подорожі;
- expansessum для визначення сумарних витрат на подорож за цим планом;
- payedsum для визначення сплачених витрат на подорож за планом;
- chosen для визначення вибору даного плану для реалізації;
- createdat для дати і часу створення плану подорожі.

Всі складові подорожі можуть обговорюватися між користувачами, зокрема експертом і мандрівником, двома мандрівниками між собою. В основі цього обговорення завжди самі повідомлення, але далі до них можуть бути додані певні складові планів подорожей для уточнення, чого саме стосується те чи інше питання або відповідь. Безпосередньо самі обговорення зведені в таблицю, що має назву Discussion і поля:

- id для ідентифікації обговорення;
- user1 для визначення користувача, який створив це обговорення (повідомлення): зовнішній ключ, пов'язує дану таблицю з таблицею User;
- user2 для визначення користувача, якому адресовано це обговорення: зовнішній ключ, пов'язує дану таблицю з таблицею User;
- text для тексту обговорення;
- createdat для дати і часу створення цього обговорення;
- private для визначення того, чи є це повідомлення приватним між користувачами, чи його може бачити будь-який користувач, який має доступ до плану подорожі.

Рекомендації місць є джерелом для подальшого визначення подорожей інших мандрівників. Вони винесені в таблицю, що має назву Recommendation і поля:

- id для ідентифікації рекомендації місця;
- traveler для визначення мандрівника, який створив рекомендацію: зовнішній ключ, пов'язує дану таблицю з таблицею Traveler;
- point для визначення місця, якого стосується рекомендація: зовнішній ключ, пов'язує дану таблицю з таблицею Point;

- plus для визначення того, чи є це рекомендацією, чи навпаки це негативний відгук;

- details для детального опису рекомендації;

- createdat для дати і часу створення рекомендації.

Відгуки між користувачами, тобто мандрівника про експерта, який допомагав йому створити план подорожі, або експерта про мандрівника, якому він цей план створював, винесені в таблицю, що має назву Feedback і поля:

- id для ідентифікації відгуку;

- planner для визначення мандрівника з відгуку: зовнішній ключ, пов'язує дану таблицю з таблицею Planner;

- traveler для визначення експерта з відгуку: зовнішній ключ, пов'язує дану таблицю з таблицею Traveler;

- mark для оцінки відгуку;

- text для тексту відгуку;

- way для напрямку відгуку: мандрівник про експерта або експерт про мандрівника;

- createdat для дати і часу створення відгуку.

Надання доступу до планів подорожі групі людей виконується через таблицю, що має назву JourneyShare і поля:

- journey для визначення подорожі, до якої надано доступ мандрівнику: зовнішній ключ, пов'язує дану таблицю з таблицею Journey;

- traveler для визначення мандрівника, якому надано доступ до подорожі: зовнішній ключ, пов'язує дану таблицю з таблицею Traveler.

Ця таблиця зв'язує між собою таблиці Traveler і Journey, реалізуючи зв'язок типу «багато-до-багатьох», тому використовується для ідентифікації записів ключ, який складається з зовнішніх ключів таблиць.

Якщо з подорожжю працює група людей, то вони можуть приймати рішення щодо подорожі спільно, обираючи план, який вони вважають

кращим. Для фіксації голосів, відданих за план мандрівником з групи, створено таблицю, що має назву JourneyPlanVote і поля:

- journeyplan для визначення плану подорожі, за який віддав голос мандрівник: зовнішній ключ, пов'язує дану таблицю з таблицею Journeyplan;
- traveler для визначення мандрівника, який віддав цей голос: зовнішній ключ, пов'язує дану таблицю з таблицею Traveler.

Кожна ділянка маршруту подорожі має бути подолана, використовуючи деякий вид транспорту. Першим варіантом є публічний транспорт (залізниця, літаки, автобуси), які переміщуються за цим маршрутом і дозволяють переміститися відповідним чином. Дані про подолання ділянок маршруту в подорожі зведені в таблицю, що має назву RouteSectionPublictransport і поля:

- id для ідентифікації транспорту, який використовується для подолання ділянки маршруту подорожі;
- routesection для визначення ділянки маршруту подорожі: зовнішній ключ, пов'язує дану таблицю з таблицею Routesection;
- departureat для дати і часу відправлення транспорту;
- departurepoint для визначення місця, з якого відправляється транспорт (це не населений пункт, а наприклад, аеропорт, вокзал): зовнішній ключ, пов'язує дану таблицю з таблицею Point;
- arriveat для дати і часу прибуття транспорту;
- arrivepoint для визначення місця, до якого прибуває транспорт (це знову ж таки не населений пункт, а наприклад, аеропорт, вокзал): зовнішній ключ, пов'язує дану таблицю з таблицею Point;
- transportcompany для визначення компанії, яка виконує перевезення: зовнішній ключ, пов'язує дану таблицю з таблицею Transportcompany;
- note для визначення примітки до цього перевезення;
- ready для визначення того, чи готове це заплановане перевезення для придбання квитка;

- `transportnumber` для номеру маршруту або польоту, за яким здійснюється перевезення;
- `ticket` для файлу квитка;
- `expansessum` для запланованої вартості цієї позиції;
- `payedsun` для сплаченої вартості за перевезення;
- `added` для визначення необхідності додавати до підсумкового результату відповідної статті витрат або відсутності такої необхідності.

Компанії, які займаються перевезеннями, зведені в таблицю, що має назву `Transportcompany` і поля:

- `id` для ідентифікації транспортної компанії;
- `caption` для назви компанії;
- `site` для адреси вебсайта компанії;
- `transporttype` для визначення типу транспортування: залізниця, літак, автобус, приватне перевезення.

Ще одним варіантом переміщення є прокат автомобіля. Прокатні компанії зведені в таблицю, що має назву `Rentalcompany` і поля:

- `id` для ідентифікації прокатної компанії;
- `caption` для назви прокатної компанії;
- `site` для адреси вебсайта компанії.

Бронювання прокату автомобіля для плану подорожі зведені в таблицю, що має назву `Car` і поля:

- `id` для ідентифікації бронювання прокату автомобіля;
- `journeuplan` для визначення плану подорожі, для якого має бути взято або взято на прокат цей автомобіль: зовнішній ключ, пов'язує дану таблицю з таблицею `Journeuplan`;

- `rentalcompany` для визначення прокатної компанії, у якої було взято автомобіль: зовнішній ключ, пов'язує дану таблицю з таблицею `Rentalcompany`;

- `mark` для визначення виробника автомобіля, який взято або треба взяти на прокат;

– model для визначення моделі автомобіля, який взято або треба взяти на прокат;

– departureat для дати і часу початку прокату;

– arriveat для дати і часу закінчення прокату;

– departurepoint для визначення місця, з якого потрібно забрати автомобіль (пункт прокату): зовнішній ключ, пов'язує дану таблицю з таблицею Point;

– arrivepoint для визначення місця, до якого потрібно повернути автомобіль (пункт прокату): зовнішній ключ, пов'язує дану таблицю з таблицею Point;

– bookingnumber для номеру бронювання;

– expansessum для запланованої суми витрат на прокат;

– payedsum для сплаченої суми за прокат;

– submitted для підтвердження необхідності виконувати бронювання прокату;

– done для підтвердження виконання бронювання прокату автомобіля;

– booking для файлу з бронюванням, якщо такий файл надається;

– added для визначення необхідності додавати до підсумкового результату відповідної статті витрат або відсутності такої необхідності.

Сервіси бронювання житла зведені в таблицю, що має назву Service і поля:

– id для ідентифікації сервісу бронювання житла;

– caption для назви сервісу;

– site для адреси вебсайту сервісу.

Самі бронювання житла зведені в таблицю, що має назву Accommodation і поля:

– id для ідентифікації заброньованого для проживання в подорожі житла;

– `journeyplanpoint` для визначення місця з плану подорожі, відносно якого зроблено або має бути зроблено це бронювання: зовнішній ключ, пов’язує дану таблицю з таблицею `JourneyplanPoint`;

- `details` для деталей проживання;
- `checkin` для визначення часу і дати початку проживання;
- `checkout` для визначення часу і дати завершення проживання;
- `number` для визначення номеру бронювання;
- `service` для визначення сервісу, з допомогою якого зроблено бронювання: зовнішній ключ, пов’язує дану таблицю з таблицею `Service`;
- `expansessum` для запланованої суми витрат на це житло;
- `payedsum` для сплаченої суми за це житло;
- `booking` для файлу бронювання;
- `address` для визначення адреси проживання (для квартири);
- `added` для визначення необхідності додавати до підсумкового результату відповідної статті витрат або відсутності такої необхідності.

Оскільки обговорення може бути пов’язано з різними ресурсами, а щодо одного ресурсу може виконуватися декілька обговорень, то створено зв’язуючі таблиці для цього.

Зв’язок між планом подорожі та обговоренням зведений в таблицю `JourneyPlanDiscussion` з полями:

- `discussion` для визначення обговорення: зовнішній ключ, пов’язує дану таблицю з таблицею `Discussion`;
- `journeyplan` для визначення плану подорожі: зовнішній ключ, пов’язує дану таблицю з таблицею `Journeyplan`.

Зв’язок між конкретним місцем з плану подорожі та обговоренням зведений в таблицю `JourneyplanPointDiscussion` з полями:

- `discussion` для визначення обговорення: зовнішній ключ, пов’язує дану таблицю з таблицею `Discussion`;

– `journeyplanpoint` для визначення конкретного місця з плану подорожі: зовнішній ключ, пов'язує дану таблицю з таблицею `JourneyplanPoint`.

Зв'язок між ділянкою маршруту та обговоренням зведений в таблицю `RouteSectionDiscussion` з полями:

- `discussion` для визначення обговорення: зовнішній ключ, пов'язує дану таблицю з таблицею `Discussion`;
- `routeSection` для визначення ділянки маршруту: зовнішній ключ, пов'язує дану таблицю з таблицею `RouteSection`.

Зв'язок між витратою та обговоренням зведений в таблицю `ExpenseDiscussion` з полями:

- `discussion` для визначення обговорення: зовнішній ключ, пов'язує дану таблицю з таблицею `Discussion`;
- `expense` для визначення витрати: зовнішній ключ, пов'язує дану таблицю з таблицею `Expense`.

Зв'язок між проживанням та обговоренням зведений в таблицю `AccommodationDiscussion` з полями:

- `discussion` для визначення обговорення: зовнішній ключ, пов'язує дану таблицю з таблицею `Discussion`;
- `accommodation` для визначення бронювання житла: зовнішній ключ, пов'язує дану таблицю з таблицею `Accommodation`.

Зв'язок між прокатом автомобіля та обговоренням зведений в таблицю `CarDiscussion` з полями:

- `discussion` для визначення обговорення: зовнішній ключ, пов'язує дану таблицю з таблицею `Discussion`;
- `car` для визначення прокату автомобіля: зовнішній ключ, пов'язує дану таблицю з таблицею `Plannerservice`.

Зв'язок між подоланням ділянки маршруту з плану подорожі певним транспортом та обговоренням зведений в таблицю `RouteSectionPublictransportDiscussion` з полями:

- discussion для визначення обговорення: зовнішній ключ, пов’язує дану таблицю з таблицею Discussion;
- routesectionpublictransport для визначення транспорту для подолання ділянки маршруту з плану подорожі: зовнішній ключ, пов’язує дану таблицю з таблицею RoutesectionPublictransport (рис. 2.2).

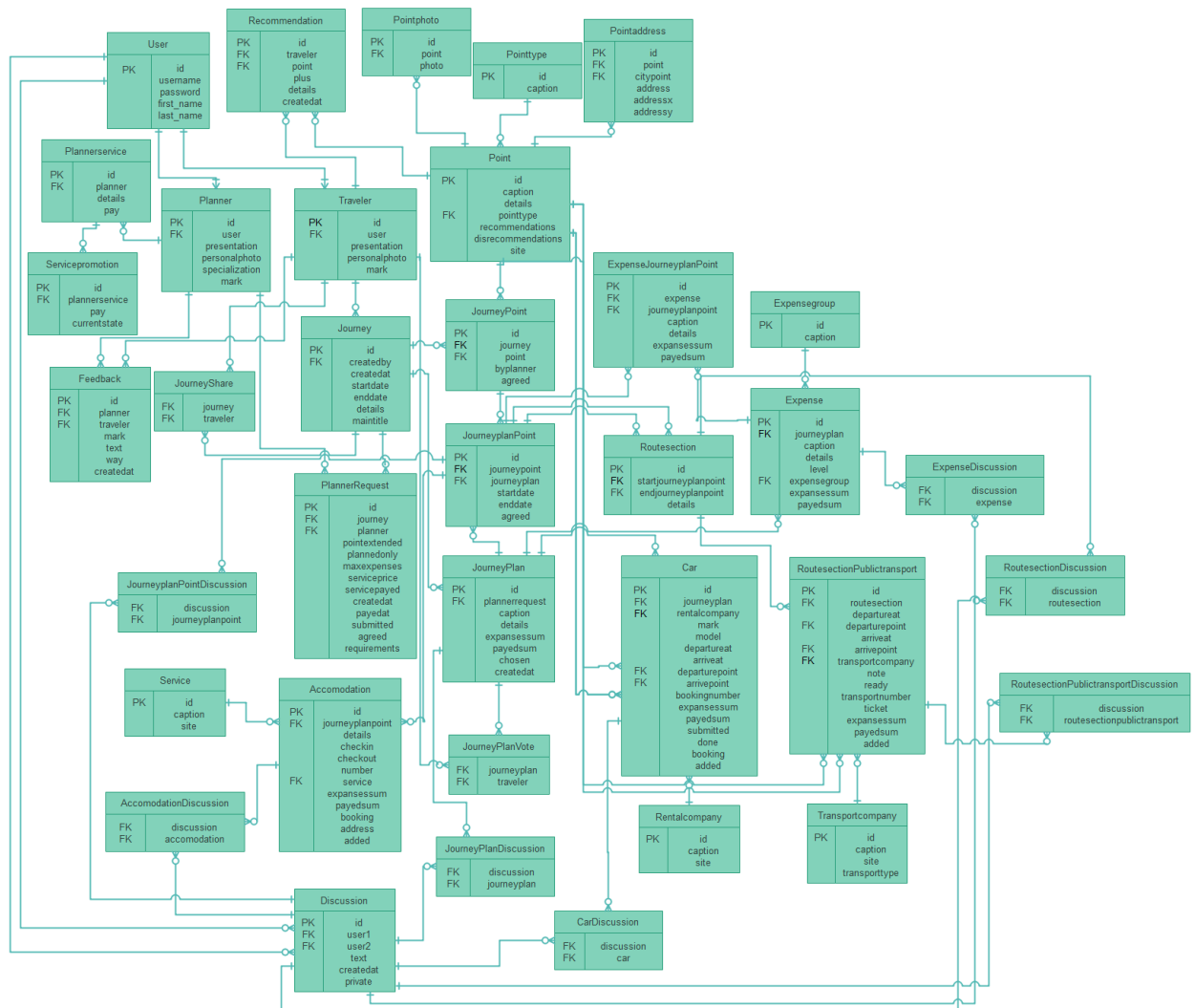


Рисунок 2.2 – Схема БД

Окремі витрати за окремим місцями з подорожі (наприклад, витрати в ресторані, витрати на екскурсію) зведені в таблицю, що має назву ExpenseJourneyplanPoint і поля:

- id для ідентифікації витрат за місцем подорожі;

- `expense` для визначення витрати на подорож, до якої належить ця витрата: зовнішній ключ, пов'язує дану таблицю з таблицею `Expense`;
- `journeyplanpoint` для визначення місця з подорожі, до якого належить ця витрата: зовнішній ключ, пов'язує дану таблицю з таблицею `JourneyplanPoint`;
- `caption` для назви витрати;
- `details` для детального опису витрати;
- `expansessum` для запланованої суми витрат;
- `payedsun` для сплаченої суми витрат.

2.4 Висновки за розділом 2

Вебсистему має бути створено, застосовуючи мову Python з фреймворком Django і системою MySQL для роботи з БД. При моделюванні сценаріїв роботи мандрівника та експерта з системою, створенні структури БД визначено структуру шарів для роботи з планами подорожей. Ці шари охоплюють саму подорож, запит за подорожжю до експерта, план подорожі, а далі відповідно місця у плані подорожі, маршрут подорожі, проживання, прокати автомобілів, переміщення публічним транспортом, що разом об'єднується через статті витрат.

3 РЕАЛІЗАЦІЯ ПРОГРАМИ

3.1 Структура класів

Використання Django для розробки визначає патерн, який використовується для побудови архітектури. Це Model-View-Template (MVT) [11]. Цей патерн разом з принципами Django визначає класи моделей, структура яких відповідає структурі БД. Наступними складовими в MVT є уявлення і шаблони. Вони формують виконання тих функцій, які мають реалізовуватися вебсистемою. Фактично кожне уявлення має відповідний шаблон, тому структура класів уявлень (рис. 3.1) є ключовою для побудови системи.

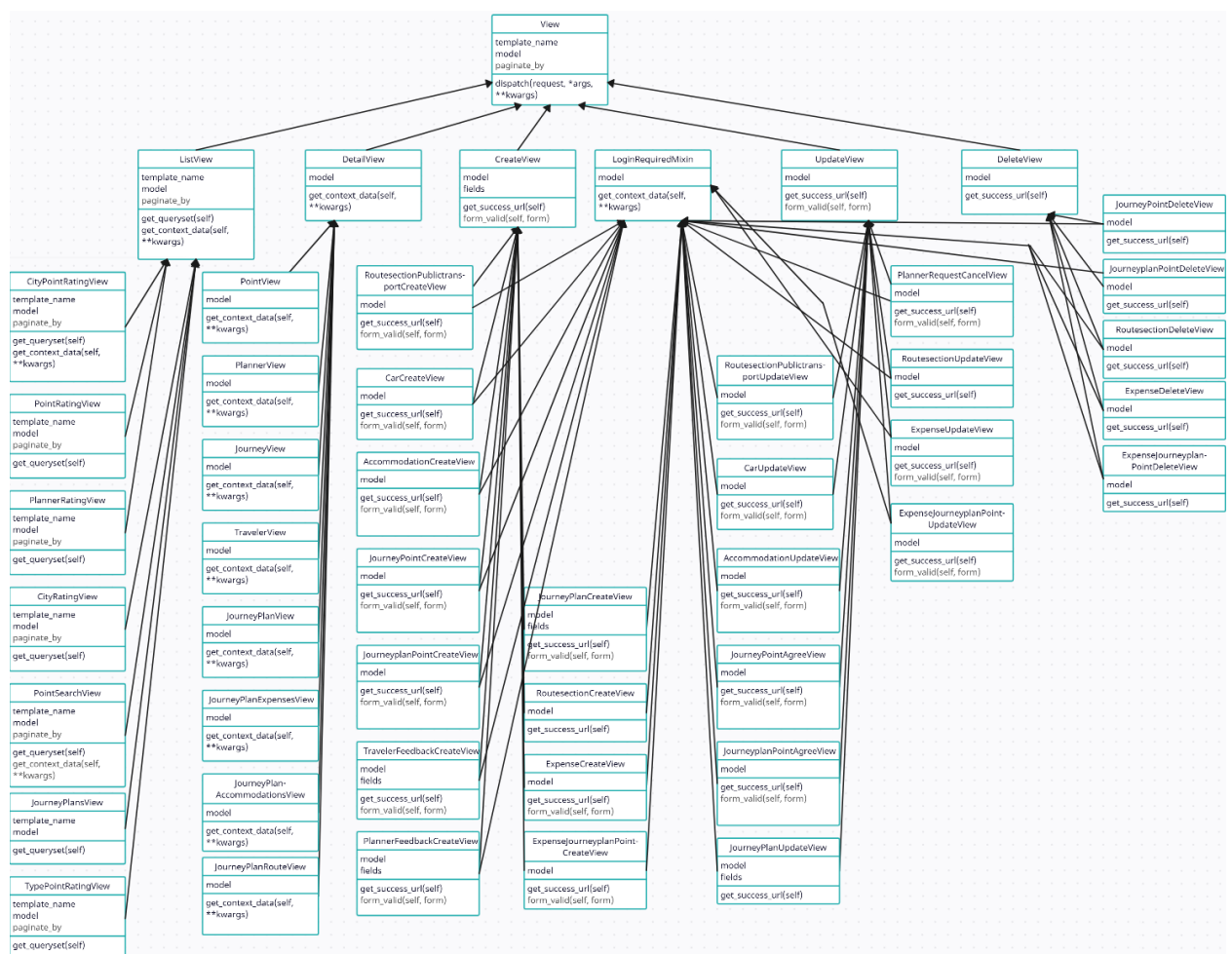


Рисунок 3.1 – Діаграма класів спільних для експерта та мандрівника

Використовуючи функціональні вимоги, кожен з них забезпечено класом або функцією у складі уявлень або декількома уявленнями, як наприклад, для забезпечення сплати за принципами MVT. Уявлення представляється об'єднанням безпосередньо класу або функції уявлення та шаблону вебсторінки. Основну кількість уявлень створено через наслідування класів, які надаються Django для забезпечення відповідної типової роботи, що в підсумку обумовило те, що нижче представлено основні класи, які мають певні особливості в своїй реалізації. На діаграмі класів на рис. 3.1 винесено тільки частину класів уявлень, які відповідають функціям і експерта, і мандрівника одночасно і які в структурі файлів системи винесені в один простір імен.

3.2 Програмна реалізація класів вебсистеми

Клас `CityPointRatingView` дозволяє отримати рейтинг місць у потрібному місті. Зважаючи на те, що в результаті отримується перелік, наслідування виконано від класу `ListView`. Моделлю визначено `Point`, адже вона об'єднує всі місця, в тому числі міста (або загалом населені пункти). Назва міста передається до уявлення з GET-запиту, в якому виділяється параметр `citypoint`. Основна функціональність з відбору місць до рейтингу створена в методі `get_queryset(self)`. Фактично в цьому методі виконується відбір всіх місць, які знаходяться в населеному пункті, назву якого отримано, а впорядковуються вони за зменшенням кількості рекомендацій. У підсумку формуються сторінки по 20 місць на кожній.

Клас `TypePointRatingView` відповідає за уявлення, пов'язане з формуванням рейтингу місць заданого типу. Тобто наприклад, рейтингу ресторанів, готелів тощо. Також тип місця передається за назвою у параметрі `typepoint`. Впорядкування у методі `get_queryset(self)` також реалізується за зменшенням рекомендацій.

Подібно створені зокрема класи `CityRatingView` для рейтингу міст за кількістю рекомендацій, `PointRatingView` для рейтингу всіх місць (не населених пунктів) у світі, `PlannerRatingView` для рейтингу експертів, до яких вносяться тільки 100 найкращих за балом відгуків.

Клас `PointView` забезпечує деталізацію інформації про конкретне місце. Тому для наслідування обрано клас `DetailView`, що забезпечує на номером отримання всіх даних відповідного об'єкту. Однак, виходячи з функціональності, яку має бути забезпечено, для місць важливим є те у розрізі програми, що вони додаються до подорожей. Тому на кожній такій сторінці потрібно знати, чи додавав вже користувач це місце до своїх подорожей. А якщо додавав, то потрібно знати, в які саме подорожі це виконувалось. А тому окрім об'єкта місця далі також було встановлено в методі `get_context_data` перелік подорожей, до яких має відношення, тобто на які має підтверджений запит, експерт, який авторизувався. З них окремо було визначено подорожі в результаті за ключем `journeys_saved`, де визначено такі подорожі, в яких уже міститься це місце, та подорожі, в яких ще не міститься це місце, – за ключем `journeys`. Також сформовано окремі ключі для визначення планів подорожей, у яких це місце міститься (`plans_saved`) та в яких це місце відсутнє (`plans`). Однак це стосується експерта, а не завжди звичайно користувач є експертом. Мандрівник також може використовувати це уявлення. Тому якщо користувач є мандрівником, то йому окрім переліків подорожей, до яких збережено це місце, та подорожей, до яких не збережено це місце, додано ще в ключі `recommendations` інформацію про те, чи рекомендував він раніше це місце, а в ключі `disrecommendations` – чи визначав він негативну рекомендацію для цього місця.

Клас `PlannerView` також є деталізацією, але такою, яка надає інформацію про експерта. Основна інформація відповідає об'єкту класу `Planner` відповідної моделі, але також на цій сторінці розміщаються відгуки про експерта від мандрівників. Відгуки знаходяться в результаті за ключем `feedbacks`. Але наявність тільки переліку відгуків звичайно не є останньою

інформацією. Необхідно також розуміти, чи має можливість мандрівник (якщо саме він зайшов на сторінку) створити відгук про цього експерта. Для цього виділяються відповідні дані про запити до цього експерта. Якщо вони були, при чому стосувалися подорожі, автором якої є мандрівник, який авторизувався, то в ключі `for_review` міститься істинне значення, а в шаблоні сторінки виводиться кнопка з можливістю створення відгуку. Якщо значення хибне, то кнопка не виводиться. Те саме, але для мандрівника виконується за допомогою класу уявлення `TravelerView`.

Також з деталізацією пов'язаний клас `JourneyView`, але він має ряд особливої інформації. Ця інформація надається тільки користувачу, який є мандрівником, що створив цю подорож або якому надано доступ до подорожі, або експерту, який має підтверджену заявку на роботу з цією подорожжю. Якщо це так, то за подорожжю виділяються:

- всі плани подорожі, впорядковуючи за датою створення з найновішої спочатку (ключ `plans`);
- всі взяті або заплановані для прокату автомобілі, що стосуються обраного для реалізації плану подорожі (ключ `cars`);
- всі заброньовані переміщення за обраним планом подорожі (ключ `transport`);
- всі заброньовані проживання за обраним планом подорожі (ключ `accommodation`);
- всі ділянки маршруту за обраним планом подорожі (ключ `routes`);
- всі запити до експертів за цією подорожжю (ключ `requests`);
- всі статті витрат за обраним планом подорожі (ключ `expenses`);
- всі місця, які було заплановано для відвідування, за обраним планом подорожі (ключ `points`).

Клас `PointSearchView` забезпечує пошук місць і тому наслідує клас `ListView`. Результати також виводяться по 20 на сторінці. У пошуку використовується ряд параметрів: тип місця (параметр `typepoint`), назва міста (населеного пункту), в якому знаходиться місце (параметр `city`), широта

координати адреси, відносно якої виконується пошук (параметр `addressx`), довгота координати адреси, відносно якої виконується пошук (параметр `addressy`), радіус пошуку навколо цієї адреси в кілометрах (параметр `radius`). Відповідно під час пошуку обчислюється відстань кожного місця в кілометрах від заданого центру на основі пакету `geopy`. Всі місця, віддалені більше ніж на заданий радіус від центру, відкидаються.

Клас `RouteSectionPublictransportCreateView` забезпечує додавання нового бронювання переміщення за маршрутом подорожі до плану. Тому створено цей клас, наслідуючи клас, що вимагає авторизації, `LoginRequiredMixin` та клас `CreateView`, що підтримує створення об'єкту в даному випадку за моделлю `RouteSectionPublictransport`. Це дозволило визначити тільки метод `get_success_url` для формування посилання, на яке перенаправляється користувач після успішного внесення даних, та метод `form_valid`, який необхідний був для того, щоб перерахувати вартість витрат. Вартість витрат (як запланованих, так і сплачених) за статтею найвищого рівня, тобто узагальнюючою, з номером групи 0, оскільки це транспортні витрати, збільшується на задані значення в формі при створенні запису про це бронювання. Також оновлюється автоматично і вартість як запланованих, так і сплачених витрат за всім планом подорожі.

Клас `RouteSectionPublictransportUpdateView` забезпечує редагування вже створеного бронювання переміщення за маршрутом подорожі за планом. Тому створено цей клас, наслідуючи клас, що вимагає авторизації, `LoginRequiredMixin` та клас `UpdateView`, що підтримує редагування об'єкту. Це дозволило визначити тільки метод `get_success_url` для формування посилання, на яке перенаправляється користувач після успішного змінювання даних, та метод `form_valid`, який необхідний був для того, щоб перерахувати вартість витрат. Вартість витрат (як запланованих, так і сплачених) за статтею найвищого рівня, тобто узагальнюючою, з номером групи 0, оскільки це транспортні витрати, змінюється на величину зміни у відповідності з формою. Для цього береться значення з форми і витягається

значення з БД за цим бронюванням. Так обчислюється різниця, після чого вона використовується в обчисленні нового значення. Також оновлюється автоматично і вартість як запланованих, так і сплачених витрат за всім планом подорожі.

Подібно організовано класи `CarCreateView`, `CarUpdateView`, `AccommodationCreateView`, `AccommodationUpdateView`.

Клас `JourneyPointCreateView` також забезпечує додавання даних, але вже місця до подорожі. У його методі `form_valid` відбувається перевірка стосовно того, чи створював цей мандрівник цю подорож або чи має експерт підтверджений запит на виконання послуг (якщо відповідно користувачем є експерт, який авторизувався). Якщо це так, то вносяться дані про те, чи було внесено місце планувальником, чи мандрівником. Якщо мандрівником, то одразу зазначається, що це місце погоджене.

Клас `JourneyPointAgreeView` забезпечує уявлення підтвердження місця в подорожі. Оскільки це внесення змін, то наслідувався клас `UpdateView`. Тут так само, як в попередньому класі, реалізується перевірка прав. Після цього за наявності таких прав та у випадку, якщо це саме мандрівник, робиться відмітка, що місце у подорожі погоджено.

Клас `JourneyPointDeleteView` забезпечує вилучення помилково доданого місця до подорожі, а тому наслідує клас `DeleteView`. Тому єдине, що для нього реалізовано, – це метод `get_success_url`, який формує посилання для перенаправлення на сторінку подорожі після вилучення.

Клас `JourneyPlanView` забезпечує інформацію про план подорожі. Ця інформація надається тільки користувачу, який є мандрівником, що створив цю подорож або якому надано доступ до подорожі, або експерту, який має підтверджену заявку на роботу з цією подорожжю. Якщо це так, то за планом подорожі окрім відповідного об'єкта класу `JourneyPlan` виділяються:

- всі взяті або заплановані для прокату автомобілі, що стосуються плану подорожі (ключ `cars`), а також усі обговорення, які стосуються цих автомобілів;

- всі заброньовані переміщення за планом подорожі (ключ `transport`), а також усі обговорення, які стосуються цих бронювань;
- всі заброньовані проживання за планом подорожі (ключ `accommodation`), а також усі обговорення, які стосуються цих бронювань;
- всі ділянки маршруту за планом подорожі (ключ `routes`), а також усі обговорення, які стосуються цих ділянок;
- всі запити до експертів за цією подорожжю (ключ `requests`), до якої належить план;
- всі статті витрат за планом подорожі (ключ `expenses`), а також усі обговорення, які стосуються цих витрат;
- всі місця, які було заплановано для відвідування, за планом подорожі (ключ `points`).

Керування обговореннями стосується і операцій створення, і перегляду переліків. Тому реалізацію виконано через уявлення на основі функції `discussions`. У цій функції спочатку для однойменного ключа `planners_direct_list` формується перелік всіх експертів, які мають заявки відносно подорожей, створених мандрівником, який авторизувався. За кожним таким експертом інформація про нього вноситься за ключем `planner`, а перелік усіх повідомлень вноситься в ключ `discussions`. За ключем `planners_shared_list` формується перелік усіх експертів, які мають підтверджені заявки стосовно подорожей, які створив мандрівник, що авторизувався в системі, та які не входять в перший перелік. За тими самими ключами вносяться дані самого експерта та перелік повідомлень. Також за ключем `travelers_shared` формується перелік усіх мандрівників, з якими має повідомлення мандрівник, який авторизувався. Окрім переліку контактів та повідомлень за контактами в обговорення також вноситься форма для створення повідомлення, яка підтримується окремим класом `DiscussionForm`.

Клас `TravelerFeedbackCreateView` забезпечує створення відгуку мандрівником про експерта. У методі `form_valid` підтримується оновлення оцінки експерта при збереженні відгуку. Для цього рахується сума всіх

оцінок за цим експертом, а далі виконується ділення цієї суми на кількість. Отримане число вноситься в оцінку експерта. Подібні дії для мандрівника виконуються в класі `PlannerFeedbackCreateView`.

3.3 Висновки за розділом 3

Завдання роботи щодо реалізації вебдодатку підтримки планів подорожей було повністю виконано. Описано структуру класів згідно з патерном MVT, правила наслідування від класів Django, логіку виконання основних дій в основних методах цих класів з акцентом на спільних класах для мандрівника та експерта.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма призначена для надання мандрівникам можливості знайти експерта, який здатен створити для них план подорожі, організувати бронювання необхідних послуг, а також для надання мандрівникам можливості керувати планом подорожі. Експерти можуть також керувати планами подорожей, вносити необхідні дані, формувати плани, бюджети подорожей.

4.2 Умови виконання програми

Умови виконання програми включають встановлення інтерпретатора мови Python, забезпечення доступу до системи MySQL для керування БД та наступні апаратні вимоги:

- процесор, мінімальна конфігурація: Intel Core 2 або подібний;
- оперативна пам'ять з доступом не менше ніж до 4 Гб;
- жорсткий диск з доступним обсягом не менше ніж 100 Гб.

4.3 Виконання і тестування програми

Тестування забезпечувалось для перевірки результативності виконання вимог, включаючи всі сценарії роботи експерта і мандрівника та браузери Chrome і Microsoft Edge. Це дозволило перевірити і коректність роботи кожної функції, і відображення інтерфейсів у різних браузерах. На основі отриманих станів було створено скріншоти, які використано в подальшому описі логіки роботи.

Для будь-якого користувача-клієнта звернення до програми є максимально простим: потрібно мати браузер і знати адресу вебсайта, за яким знаходиться вебсистема.

Мандрівник має 5 основних пунктів меню та 1 кнопку. Кнопка забезпечує вихід, якщо мандрівник авторизувався, або авторизацію, якщо мандрівник ще не авторизувався. Сторінка авторизації дозволяє перейти до реєстрації.

5 пунктів меню містять роботу з експертами, що включає у вигляді випадających підпунктів пошук експертів та рейтинг експертів, роботу з місцями, що включає підпункти для пошук місць, рейтингів місць за містом, загальною або за типом місця, роботу з подорожами, що включає створення нової подорожі, з чого розпочинається вся процедура роботи над всіма складовими подорожі, плану тощо, перегляд авторських (своїх) подорожей та перегляд подорожей, до яких мандрівнику було надано доступ, роботу з обговореннями та роботу з персональною інформацією (пункт «Персона»). Робота з персональною інформацією включає окрім перегляду з можливістю переходу до редагування даних про себе ще й перегляд всіх відгуків.

Обговорення містить перелік всіх експертів, з якими спілкувався або які стосувалися мандрівника через подорожі, до яких йому було надано доступ. З цього переліку можна обрати потрібного експерта, тоді відобразиться перелік повідомлень у обговоренні з ним та форма для внесення нового повідомлення з можливістю вибору місця з плану подорожі, бронювання, яких стосується це обговорення.

Під час створення подорожі, роботи з подорожжю одним з важливих елементів є пошук місць. Вони використовуються як складові подорожі і конкретного плану. Пошук може скористатися сам мандрівник, щоб додати деякі місця, які він хоче обов'язково відвідати. Пошуком також може скористатися експерт у процесі планування подорожі. Окрім пошуку можна переглянути рейтинги місць. Одним з варіантів місць є самі міста (населені пункти). Можна скористатися загальним рейтингом (рис. 4.1) для цього. За кожним місцем показується його назва, кількість рекомендацій та негативних думок, базова фотографія. Щоб додати місце до подорожі, потрібно відкрити сторінку цього місця, де будуть показані всі фотографії, рекомендації, а

також перелік подорожей, куди це місце може додати мандрівник або експерт.

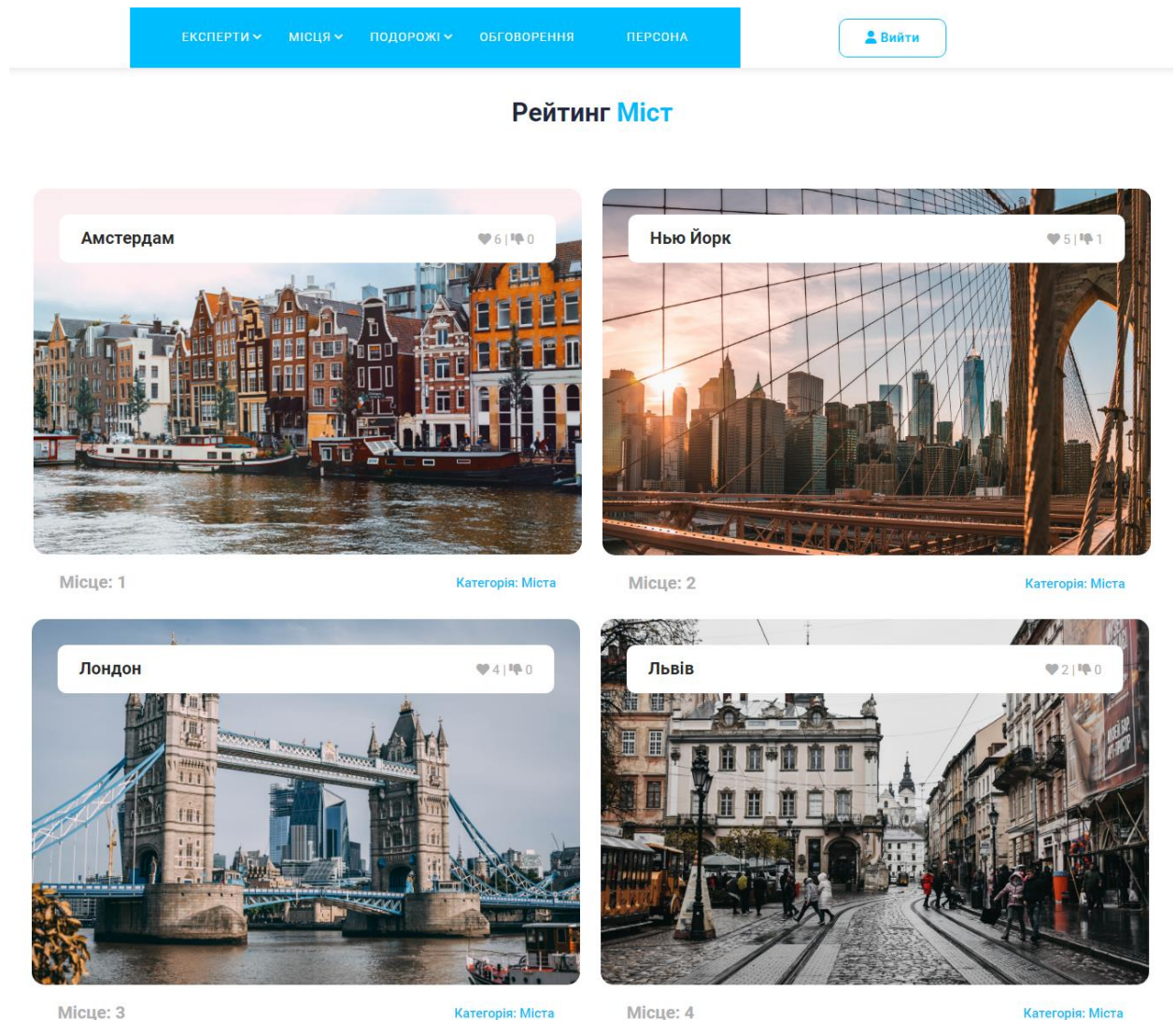


Рисунок 4.1 – Рейтинг міст (населених пунктів)

Подібним чином формуються всі рейтинги, які були описані в меню. Для мандрівника важливо сформулювати подорож, обрати якісь місця до неї, якщо він вважає за потрібне. Але не менш важливим для мандрівника, якщо він не виконує планування самостійно, є пошук експерта, якому б він довірив складання плану його подорожі. Пошук експертів включає визначення логіну експерта, якщо його знає мандрівник і шукає конкретного експерта, подорож зі всіх подорожей, які створив мандрівник і які ще не відбулись (за подорожжю визначаються міста, а експерти, які їх вказали в спеціалізації

своїй, вказуються в результатах раніше), максимальна вартість планування подорожі, яку готовий взяти на себе мандрівник, мінімальна оцінка експерта.

Результати пошуку (рис. 4.2) містять ім'я експерта з підписом логіну знизу, середню оцінку, фотографію, можливість перейти на сторінку експерта, де окрім відгуків відображаються також ціни на його послуги. Кнопка «Зробити заявку» дозволяє створити заявку на останню додану подорож мандрівника (за замовчуванням) або на наступній сторінці обрати одну з інших подорожей мандрівника.

ЕКСПЕРТИ ▼ МІСЦЯ ▼ ПОДРОЖІ ▼ ОБГОВОРЕННЯ ПЕРСОНА Вийти

Логін: введіть логін, якщо маєте інформацію... Подорож: Паризька зустріч ▼ Максимальна вартість: 10000 Мінімальна оцінка: 3 ▼

Шукати Експертів

ПОШУК

Результати Пошуку Експертів

Рейтинг Експертів

Експерт	Оцінка	Логін	Дія
Вероніка Мотренко	5.0	motrenkov	Зробити Заявку
Артур Варданян	4.8	arturch	Зробити Заявку
Іван Шатренко	4.5	ivansha	Зробити Заявку
Олександр Чистов	4.0	chistik	Зробити Заявку

Рисунок 4.2 – Пошук експертів

За створеною заявкою потім експерт має надати підтвердження або скасувати її. Після цього мандрівнику на сторінці подорожі відображається снопка сплатити для переходу до оплати. Після сплати експерт починає працювати з подорожжю. Вносити зміни в саму заявку вже тоді не можна.

Сторінка подорожі для її автора (рис. 4.3) містить цілий ряд інформації. Згори знаходиться фотографія першого міста з подорожі, інформація про авторство, строки подорожі, загальний опис, учасники (автор і ті, кому автор надав доступ). Кнопка «Надати доступ» дозволяє додати іншого мандрівника за його логіном до подорожі. Він тоді розглядається як учасник подорожі. Кнопка «Редагувати» дозволяє внести зміни в загальні дані подорожі.

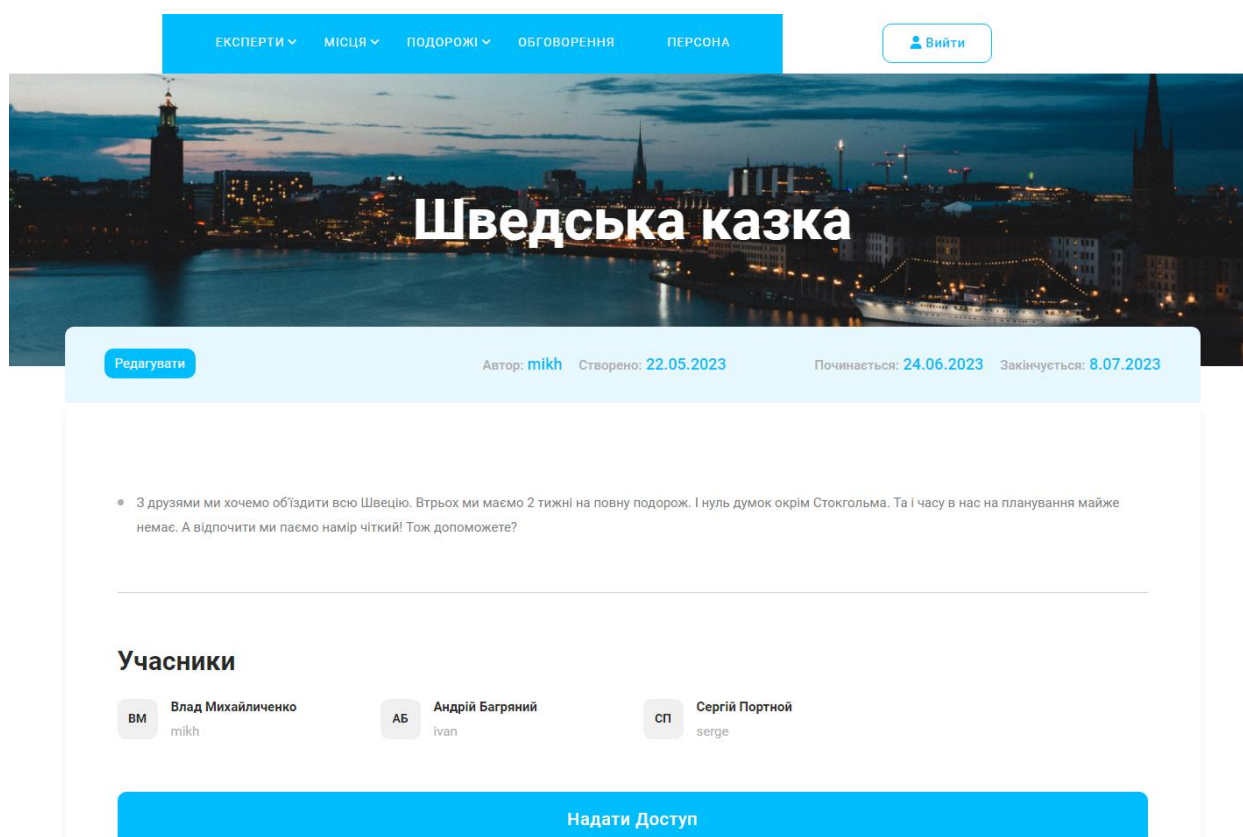


Рисунок 4.3 – Перегляд інформації про авторську подорож

Далі на сторінці знаходяться заявки до експертів на планування цієї подорожі (рис. 4.4). Кнопка «Створити заявку» дозволяє за логіном звідси обрати потрібного експерта і визначити далі заявку до нього на планування цієї подорожі. За кожною заявкою наводиться ряд даних. Якщо заявка ще не підтверджена у виконанні, то її можна також редагувати або скасовувати. Експерт відповідно з такого переліку всіх заявок до нього може підтверджувати, скасовувати їх, створювати плани за заявкою.

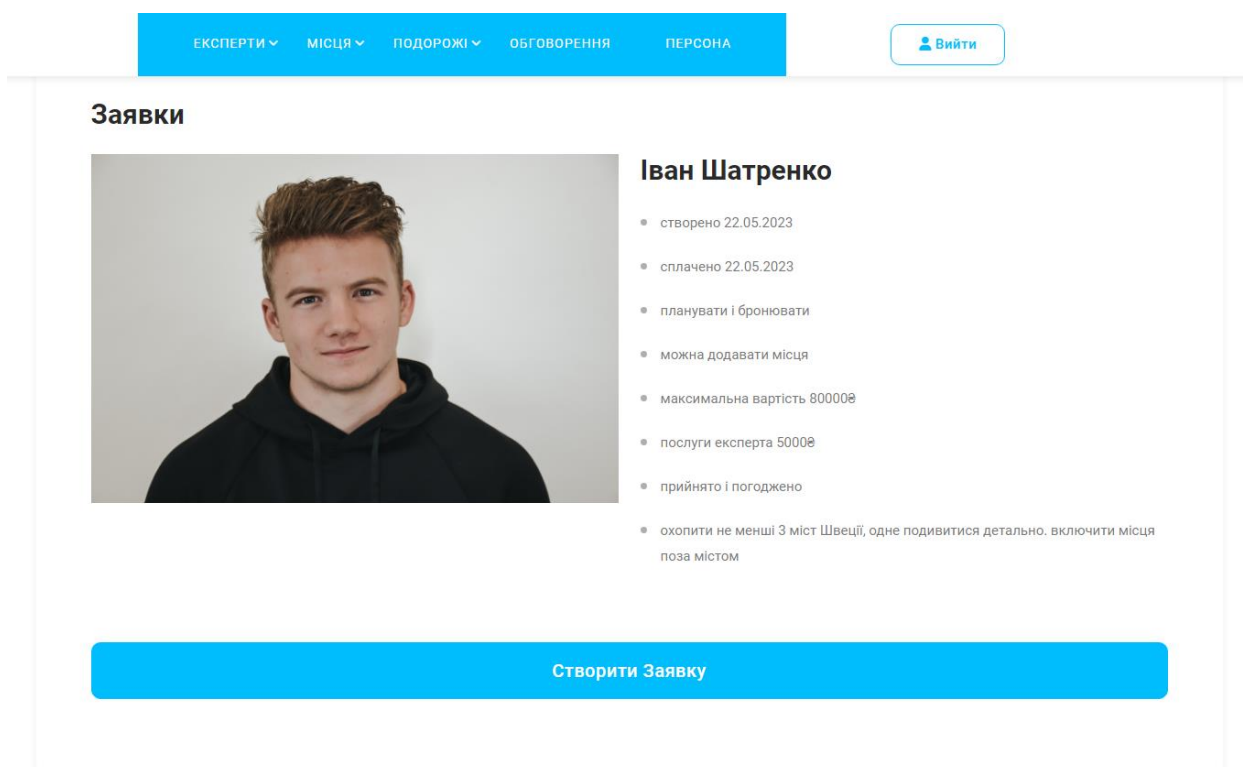


Рисунок 4.4 – Перегляд створених заявок на авторську подорож

Нижче на цій сторінці міститься перелік планів за цією подорожжю. І мандрівник, і експерт можуть створювати плани. Подальша робота з планом відбувається на сторінці плану. Але після того, як один з планів буде затверджено (шляхом голосування всіх учасників або шляхом вибору натисканням на кнопку під одним з планів автором, якщо у мандрівці тільки один учасник), на сторінці подорожі відображається його основна інформація, тобто інформація обраного для реалізації плану подорожі.

Тому на цій сторінці нижче можуть міститися дані, які фактично є складовими сторінки плану подорожі, зокрема: ділянки маршруту подорожі, місця, які включені до плану подорожі, перелік бронювань за проживаннями, за прокатом автомобілів, за переїздами, витрат за подорожжю. Для деталізації потрібно за цими складовими відкрити окрему сторінку. За кожною цією складовою відображається поле для створення повідомлення та перелік повідомлень, якщо вони вже були створені. Тобто в обговореннях знаходяться всі повідомлення, а на окремих сторінках ті, що стосуються конкретних складових.

Відповідно для експерта основною точкою роботи є робота над планом подорожі або планами. За кожним планом він може додавати до описаних складових відповідні позиції, редагувати існуючі. Наприклад, щодо бронювань проживань за подорожжю (рис. 4.5) експерт отримує перелік дат з основними містами за всіма датами, куди він може додати бронювання проживання.

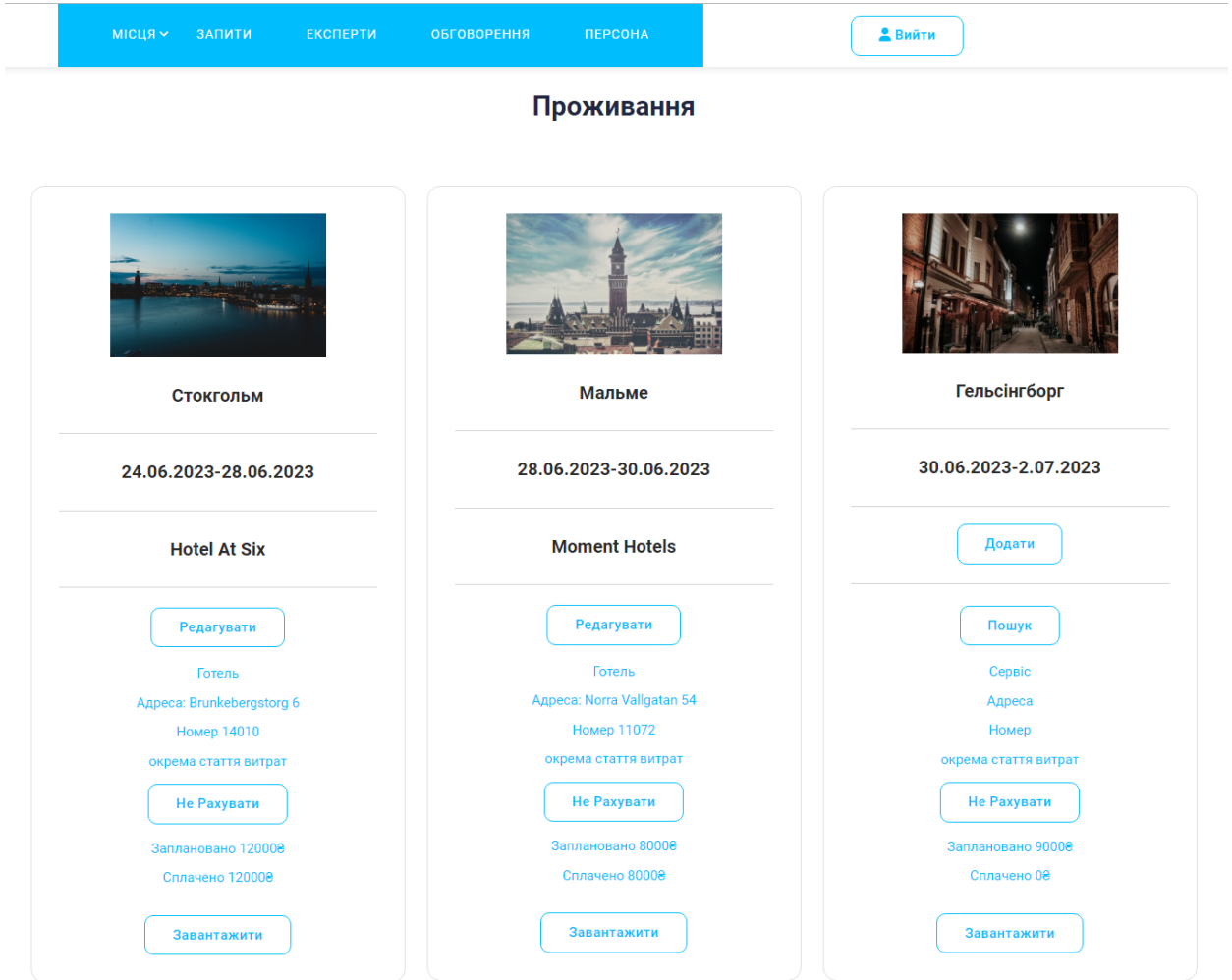


Рисунок 4.5 – Планування подорожі експертом

Місцем, в якому мандрівник буде проживати в подорожі є те, яке або включає декілька дат поспіль або те, яке на окрему дату вказано останнім. Тобто це означає, що в цьому місці мандрівник буде ночувати. Вміст цієї частини сторінки залежить від того, що було вказано в маршруті подорожі і від того, що було вказано в місцях плану подорожі. За кожною датою

виводиться місце, де мандрівник буде проживати в цю дату або дати, місце проживання (готель, квартира). Якщо місце не визначено, то його можна додати через пошук або просто введенням через відповідну форму. За кожною позицією також вказується, чи потрібно витрати за цим бронюванням окремо додавати до відповідних статей витрат, чи вони вже враховані в тих статтях. Для цього використовується кнопка «Не рахувати». Вона дозволяє перемикатися між цими варіантами. Кнопка «Завантажити» дозволяє завантажити файл бронювання, якщо він вже є.

4.4 Висновки за розділом 4

І експерт, і мандрівник можуть виконувати планування подорожі. Але вони мають і особливі засоби. Основні інструменти для кожного користувача представлені як візуально на прикладах, так і шляхом опису логіки роботи.

ВИСНОВКИ

Мандрівники, які подорожують, не використовуючи послуги туристичних компаній, все одно потребують створення плану подорожі, придбання квитків, бронювання житла. Вони можуть створити цей план або самостійно, або використовуючи послуги експертів. Тому метою роботи було розробити програмне забезпечення підтримки планів подорожей для забезпечення мандрівників відповідними засобами поза межами туристичних компаній.

Розробка мала об'єднати можливості створення планів подорожей з можливостями розділення доступу до цих планів, щоб ними можна було певним чином керувати разом групі мандрівників, та з можливостями доступу до ринку послуг експертів, які надають послуги планування, щоб вони, а не сам мандрівник, могли створити план подорожі, виконати потрібні бронювання для цього, а мандрівник міг тільки відпочивати або подорожувати за цим планом за подібною логікою до туристичних компаній, але зі значно більшою персоналізацією подорожі.

Вебсистему було створено, застосовуючи мову Python з фреймворком Django і системою MySQL для роботи з БД. При моделюванні сценаріїв роботи мандрівника та експерта з системою, створенні структури БД визначено структуру шарів для роботи з планами подорожей. Ці шари охоплюють саму подорож, запит за подорожжю до експерта, план подорожі, а далі відповідно місця у плані подорожі, маршрут подорожі, проживання, прокати автомобілів, переміщення публічним транспортом, що разом об'єднується через статті витрат.

Програмне забезпечення дозволяє знаходити експертів для планування подорожей, які можуть створювати план або плани подорожі, готувати всі необхідні бронювання для мандрівника. Також мандрівники можуть знаходити нові місця, додавати їх до структури подорожі, визначаючи таким чином вимоги до планування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Tsvetkov, T. 12 Tourism Trends That Will Shape the Travel Industry in 2022 and Beyond [Electronic resource] / T. Tsvetkov. – Access mode : <https://pro.regiondo.com/blog/tourism-trends-2018-2/>.
2. Europe Trip Planner & Multi-City Itinerary Builder [Electronic resource]. – Access mode : <https://triptile.com/>.
3. Wanderlog [Electronic resource] : travel itinerary, vacation & road trip planner. – Access mode : <https://wanderlog.com/home>.
4. Expert Itinerary Planning. Wandering Earl [Electronic resource]. – Access mode : <https://www.wanderingearl.com/expert-itinerary-planning/>.
5. Our Tours – Wild Air Travel [Electronic resource]. – Access mode : <https://www.wildairtravel.com/our-tours/>.
6. Travel Planning with Local Experts Around the World [Electronic resource]. – Access mode : <https://goaskalocal.com/>.
7. Global Game Plan [Electronic resource]. – Access mode : <https://www.globalgameplan.com/>.
8. Ernesti, J. Python 3 [Text] : The Comprehensive Guide to Hands-On Python Programming / Johannes Ernesti, Peter Kaiser. – Bonn : Rheinwerk Computing, 2022. – 1078 p.
9. Mele, A. Django 4 By Example [Text] : Build powerful and reliable Python web applications from scratch / Antonio Mele, Bob Belderbos. – Birmingham : Packt Publishing, 2022. – 766 p.
10. Snapx Photography Template [Electronic resource]. – Access mode : <https://templatemo.com/tm-576-snapx-photography>.
11. Shadhin, F. The MVT Design Pattern of Django [Text] / F. Shadhin. – Access mode : <https://python.plainenglish.io/the-mvt-design-pattern-of-django-8fd47c61f582>.

ДОДАТОК А
Технічне завдання

Вступ

Планування подорожей – це цілий процес, від якісного виконання якого залежить те, на скільки задоволеним в підсумку буде мандрівник. Для цього потрібно мати певні знання, досвід, вміти правильно об'єднувати різні складові у цілісний план. Будь-які помилки можуть призвести до того, що враження про всю подорож зіпсується, а також до проблем. Відповідно можна говорити про те, що це окремий і достатньо тривалий процес, особливо за відсутності знань і досвіду. У результаті мандрівник може мати бажання, але не мати часу все спланувати якісно. У такому випадку люди з досвідом можуть допомогти сформувати план, виконати потрібні бронювання, щоб мандрівник сконцентрувався на подорожі, на відпочинку. Саме на підтримку взаємодії між мандрівника і експертами, тобто людьми, здатними сформувати план подорожі, направлена розробка, яку заплановано в роботі.

А.1 Підстави для розробки

Завдання на дипломну кваліфікаційну роботу визначено на підставі наказу № 87 від 4 квітня 2023 р. за Національним університетом «Запорізька політехніка» за темою «Програмне забезпечення підтримки планів подорожей».

А.2 Призначення розробки

Програма призначена для надання мандрівникам можливості знайти експерта, який здатен створити для них план подорожі, організувати бронювання необхідних послуг, а також для надання мандрівникам можливості керувати планом подорожі. Експерти можуть також керувати

планами подорожей, вносити необхідні дані, формувати плани, бюджети подорожей.

А.3 Основні вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Для підтримки описаного призначення програма повинна надавати користувачам наступний набір функцій для роботи:

- пошук експертів;
- представлення рейтингу експертів;
- рекомендування місця;
- створення нової подорожі;
- редагування даних існуючої подорожі;
- представлення авторських подорожей;
- представлення доступних подорожей;
- надання доступу до подорожі мандрівнику;
- керування доступом до подорожі;
- відправлення запиту експерту за подорожжю;
- представлення запитів за подорожжю;
- представлення планів за подорожжю;
- голосування за план подорожі;
- обговорення планів подорожей;
- створення відгуку про взаємодію між експертом і мандрівником;
- визначення цін на послуги експерта;
- надання знижки на послуги експертом;
- представлення рейтингу міст;
- представлення рейтингу місць у місті;
- представлення рейтингу місць загалом;
- представлення рейтингу місць за типом;
- створення нового плану подорожі;

- редагування існуючого плану подорожі;
- визначення маршруту за планом подорожі;
- реєстрація та авторизація користувачів, як мандрівників, так і експертів;
- додавання статей витрат до плану подорожі;
- редагування статей витрат у плані подорожі;
- представлення бюджету за планом подорожі;
- представлення маршруту за планом подорожі;
- представлення проживань за планом подорожі;
- представлення запитів до експерта;
- узгодження або скасування запиту;
- узгодження або скасування статей плану подорожі;
- оплата послуг експерта;
- додавання нового бронювання проживання до плану подорожі;
- додавання нового бронювання прокату автомобіля до плану подорожі;
- додавання нового бронювання переміщення за маршрутом подорожі до плану;
- редагування існуючого бронювання проживання;
- редагування існуючого бронювання прокату автомобіля до плану подорожі;
- пошук місць;
- вилучення статті плану подорожі;
- редагування існуючого бронювання переміщення;
- представлення інформації про місце;
- представлення інформації про експерта;
- представлення інформації про мандрівника;
- представлення інформації про подорож;
- представлення інформації про план подорожі;
- додавання або вилучення місць з подорожі.

А.3.2 Вимоги до складу та параметрів технічних засобів

Апаратні вимоги до серверної сторони:

- процесор у мінімальній конфігурації Intel Core 2 або відповідний еквівалент;
 - оперативна пам'ять з доступом не менше ніж до 4 Гб;
 - жорсткий диск з бажаним доступним обсягом не менше ніж 100 Гб:
- у випадку меншого обсягу потрібно контролювати обсяг даних у БД при внесенні фотографій, при цьому в будь-якому випадку потрібний обсяг може і значно збільшуватися, що залежить від кількості користувачів, кількості та якості фотографій, які вони завантажують.

А.4 Порядок контролю та приймання

Основою щодо реалізації контролю та приймання дипломної кваліфікаційної роботи є завдання на роботу і безпосередньо дане технічне завдання. За успішного виконання цих завдань у встановлені строки робота після проходження всіх підготовчих процедур вноситься на захист перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

```

    from django.views.generic.edit import CreateView,
UpdateView, DeleteView
    from django.contrib.auth.mixins import LoginRequiredMixin
    from django.views.generic.detail import DetailView
    from django.db.models import Q
    from django.views.generic.list import ListView
    from django.urls import reverse
    from .models import Point, Pointaddress, Pointtype,
Recommendation, Planner, Traveler, Journey, PlannerRequest,
JourneyPoint, JourneyPlan, JourneyplanPoint, Feedback,
Accommodation, RoutesectionPublictransport, Car, Routesection,
Expense, Discussion
    from .forms import DiscussionForm
    import datetime
    import geopy.distance
    from django.shortcuts import redirect
    from django.db.models import Sum

    class CityPointRatingView(ListView):
        template_name = 'citypoints.html'
        model = Point
        paginate_by = 20

        def get_queryset(self):
            city_point_caption =
self.request.GET.get('citypoint')
            if city_point_caption:
                city_point = Point.objects.get(caption =
city_point_caption)
            else:
                return []
            if city_point.pointtype__id == 0:
                city_points =
Pointaddress.objects.filter(citypoint =
city_point).distinct('citypoint')
            else:
                return []
            city_points = city_points.order_by('-
recommendations')
            return city_points

        def get_context_data(self, **kwargs):
            context_points = super(CityPointRatingView,
self).get_context_data(**kwargs)
            city_point_caption =
self.request.GET.get('citypoint')

```

```

        if city_point_caption:
            city_point = Point.objects.get(caption =
city_point_caption)
            context_points ['city'] = city_point
        return context_points

class CityRatingView(ListView):
    template_name = 'citys.html'
    model = Point
    paginate_by = 20

    def get_queryset(self):
        city_points = Point.objects.get(pointtype__id = 0)
        city_points = city_points.order_by('-
recommendations')
        return city_points

class TypePointRatingView(ListView):
    template_name = 'typepoints.html'
    model = Point
    paginate_by = 20

    def get_queryset(self):
        type_point_caption =
self.request.GET.get('typepoint')
        if type_point_caption:
            type_point = Pointtype.objects.get(caption =
type_point_caption)
        else:
            return []
        type_points = Point.objects.filter(pointtype =
type_point)
        type_points = type_points.order_by('-
recommendations')
        return type_points

class PointRatingView(ListView):
    template_name = 'points.html'
    model = Point
    paginate_by = 20

    def get_queryset(self):
        points = Point.objects.filter(~Q(pointtype__id =
0))

        points = points.order_by('-recommendations')
        return points

```

```

class PointView(DetailView):
    model = Point

    def get_context_data(self, **kwargs):
        context_point = super().get_context_data(**kwargs)
        point = Point.objects.get(id = self.kwargs['pk'])
        context_point ["point"] = point
        user = self.request.user
        if user and user.is_active:
            traveler = Traveler.objects.get(user = user,
None)

            if not traveler:
                planner = Planner.objects.get(user = user,
None)

                requests =
PlannerRequest.objects.filter(planner = planner, submitted = 1)
                plans =
JourneyPlan.objects.filter(plannerrequest__in = requests)
                journeys = requests.distinct('journey')
                journeys_saved =
JourneyPoint.objects.filter(journey__in = journeys, point =
point).distinct('journey')
                journeyplans =
JourneyplanPoint.objects.filter(journeypoint__point = point,
journeyplan__in = plans, agreed = 1).distinct('journeyplan')
                context_point ["journeys_saved"] =
journeys_saved

                context_point ["journeys"] = journeys
                context_point ["plans"] = plans
                context_point ["plans_saved"] =
journeyplans

            else:
                recommendation =
Recommendation.objects.get(point = point, traveler = traveler,
plus = True, None)

                if recommendation:
                    context_point ["recommendations"] =
True

                else:
                    context_point ["recommendations"] =
False

                dis = Recommendation.objects.filter(point =
point, traveler = traveler, plus = False)
                if dis:
                    context_point ["disrecommendations"] =
True

```

```

        else:
            context_point ["disrecommendations"] =
False
            current = datetime.datetime.now()
            journeys = Journey.objects.filter(createdby
= traveler, startdate__gt = current)
            journeys_saved =
JourneyPoint.objects.filter(journey__in = journeys, point =
point).distinct('journey')
            context_point ["journeys_saved"] =
journeys_saved
            context_point ["journeys"] = journeys
            return context_point

class PlannerView(DetailView):
    model = Planner

    def get_context_data(self, **kwargs):
        context_point = super().get_context_data(**kwargs)
        planner = Planner.objects.get(id =
self.kwargs['pk'])
        context_point ["planner"] = planner
        feedbacks = Feedback.objects.filter(planner =
planner, way = False).order_by('-createdat')
        context_point ["feedbacks"] = feedbacks
        user = self.request.user
        if user and user.is_active:
            traveler = Traveler.objects.get(user = user,
None)

            if traveler:
                requests =
PlannerRequest.objects.filter(planner = planner,
journey__createdby = traveler)
                if requests:
                    context_point ["for_review"] = True
                else:
                    context_point ["for_review"] = False
            return context_point

class TravelerView(DetailView):
    model = Traveler

    def get_context_data(self, **kwargs):
        context_point = super().get_context_data(**kwargs)
        traveler = Traveler.objects.get(id =
self.kwargs['pk'])

```

```

        context_point ["traveler"] = traveler
        feedbacks = Feedback.objects.filter(traveler =
traveler, way = True).order_by('-createdat')
        context_point ["feedbacks"] = feedbacks
        user = self.request.user
        if user and user.is_active:
            planner = Planner.objects.get(user = user,
None)

            if planner:
                requests =
PlannerRequest.objects.filter(planner = planner,
journey__createdby = traveler)
                if requests:
                    context_point ["for_review"] = True
                else:
                    context_point ["for_review"] = False
            return context_point

class PlannerRatingView(ListView):
    template_name = 'planners.html'
    model = Planner
    paginate_by = 10

    def get_queryset(self):
        planners = Planner.objects.all().order_by('-
mark')[:100]
        return planners

class JourneyView(DetailView):
    model = Journey

    def get_context_data(self, **kwargs):
        context_point = super().get_context_data(**kwargs)
        journey = Journey.objects.get(id =
self.kwargs['pk']).prefetch_related('votes')
        user = self.request.user
        planner = Planner.objects.filter(user = user)
        allowed = False
        if user.is_active and planner:
            requests =
PlannerRequest.objects.filter(journey = journey, planner =
planner, submitted = 1)
            if requests:
                allowed = True
        elif user.is_active:
            traveler = Traveler.objects.filter(user = user)

```

```

        if traveler:
            if journey.createdby == traveler or
traveler in journey.shares:
                allowed = True
            if allowed:
                context_point ["journey"] = journey
                plans =
JourneyPlan.objects.filter(plannerrequest__ journey = journey,
submitted = 1).order_by('-createdat')
                context_point ["plans"] = plans
                main_plan = plans.filter(chosen = True)[0]
                cars = Car.objects.filter(journeyplan =
main_plan).order_by('departureat')
                context_point ["cars"] = cars
                transport =
RoutesectionPublictransport.objects.filter(routesection__startjo
urneyplanpoint__journeyplan = main_plan).order_by('departureat')
                context_point ["transport"] = transport
                accommodation =
Accommodation.objects.filter(journeyplanpoint__journeyplan =
main_plan).order_by('departureat')
                context_point ["accommodation"] = accommodation
                routes =
Routesection.objects.filter(startjourneyplanpoint__journeyplan =
main_plan).order_by('startdate')
                context_point ["routes"] = routes
                requests =
PlannerRequest.objects.filter(journey = journey).order_by('-
createdat')
                context_point ["requests"] = requests
                expenses = Expense.objects.filter(journeyplan =
main_plan).order_by('id')
                context_point ["expenses"] = expenses
                points =
JourneyplanPoint.objects.filter(journeyplan =
main_plan).order_by('startdate')
                context_point ["points"] = points
            return context_point

class PointSearchView(ListView):
    template_name = 'pointsearch.html'
    model = Point
    paginate_by = 20

    def get_queryset(self):
        city_caption = self.request.GET.get('city')

```

```

        points = []
        if city_caption:
            city_point = Point.objects.get(caption =
city_caption, pointtype__id = 0)
            points = Pointaddress.objects.filter(citypoint
= city_point).distinct('point')
            type_point_caption =
self.request.GET.get('typepoint')
            if type_point_caption:
                type_point = Pointtype.objects.get(caption =
type_point_caption)
                if not points:
                    points = Point.objects.filter(pointtype =
type_point)
                else:
                    points = points.filter(pointtype =
type_point)
            address_x = self.request.GET.get('addressx')
            address_y = self.request.GET.get('addressy')
            radius = self.request.GET.get('radius')
            radius = float(radius)
            addresses = Pointaddress.objects.filter(point__in =
points)
            address_point = (float(address_x),
float(address_y))
            points = points.order_by('-recommendations')
            points_in_radius = []
            for point_address in addresses:
                dist = geopy.distance.geodesic(address_point,
point_address).km <= radius
                if dist:
                    points_in_radius.append(point_address)
            return points_in_radius

    def get_context_data(self, **kwargs):
        context_points = super(PointSearchView,
self).get_context_data(**kwargs)
        type_point_caption =
self.request.GET.get('typepoint')
        if type_point_caption:
            type_point = Pointtype.objects.get(caption =
type_point_caption)
            context_points ['type'] = type_point
            city_caption = self.request.GET.get('city')
            points = []
            if city_caption:

```

```

        city_point = Point.objects.get(caption =
city_caption, pointtype__id = 0)
        context_points ['city'] = city_point
        address_x = self.request.GET.get('addressx')
        address_y = self.request.GET.get('addressy')
        radius = self.request.GET.get('radius')
        context_points ['addressx'] = address_x
        context_points ['addressy'] = address_y
        context_points ['radius'] = radius
        return context_points

```

```

class
RouteSectionPublictransportCreateView(LoginRequiredMixin,
CreateView):
    model = RouteSectionPublictransport

    def get_success_url(self):
        return reverse('journey-plan',
kwargs={'journeyplan_id': self.object.startjourneyplanpoint
routeSection.journeyplan.id})

    def form_valid(self, form):
        if form.instance.added:
            route = form.instance.routeSection
            journey_plan =
route.startjourneyplanpoint.journeyplan
            exp = Expense.objects.get(journeyplan =
journey_plan, level = 0, expensegroup__id = 0)
            exp.expansessum = exp.expansessum +
form.instance.expansessum
            exp.payedsum = exp.payedsum +
form.instance.payedsum
            exp.save()
            journey_plan.expansessum =
journey_plan.expansessum + form.instance.expansessum
            journey_plan.payedsum = journey_plan.payedsum +
form.instance.payedsum
            journey_plan.save()
            return super().form_valid(form)

```

```

class
RouteSectionPublictransportUpdateView(LoginRequiredMixin,
UpdateView):
    model = RouteSectionPublictransport

    def get_success_url(self):

```

```

        return reverse('journey-plan',
kwargs={'journeyplan_id':
self.object.startjourneyplanpoint.routesection.journeyplan.id})

    def form_valid(self, form):
        if form.instance.added:
            route = form.instance.routesection
            journey_plan =
route.startjourneyplanpoint.journeyplan
            transport =
RoutesectionPublictransport.objects.get(id = self.object.id)
            exp = Expense.objects.get(journeyplan =
journey_plan, level = 0, expensegroup__id = 0)
            exp.expansessum = exp.expansessum +
form.instance.expansessum - transport.expansessum
            exp.payedsum = exp.expansessum +
form.instance.payedsum - transport.payedsum
            exp.save()
            journey_plan.expansessum =
journey_plan.expansessum + form.instance.expansessum -
transport.expansessum
            journey_plan.payedsum = journey_plan.payedsum +
form.instance.payedsum - transport.payedsum
            journey_plan.save()
            return super().form_valid(form)

class CarCreateView(LoginRequiredMixin, CreateView):
    model = Car

    def get_success_url(self):
        return reverse('journey-plan',
kwargs={'journeyplan_id': self.object.journeyplan.id})

    def form_valid(self, form):
        if form.instance.added:
            journey_plan = form.instance.journeyplan
            exp = Expense.objects.get(journeyplan =
journey_plan, level = 0, expensegroup__id = 0)
            exp.expansessum = exp.expansessum +
form.instance.expansessum
            exp.payedsum = exp.payedsum +
form.instance.payedsum
            exp.save()
            journey_plan.expansessum =
journey_plan.expansessum + form.instance.expansessum

```

```

        journey_plan.payedsum = journey_plan.payedsum +
form.instance.payedsum
        journey_plan.save()
        return super().form_valid(form)

class CarUpdateView(LoginRequiredMixin, UpdateView):
    model = Car

    def get_success_url(self):
        return reverse('journey-plan',
kwargs={'journeyplan_id': self.object.journeyplan.id})

    def form_valid(self, form):
        if form.instance.added:
            journey_plan = form.instance.journeyplan
            car = Car.objects.get(id = self.object.id)
            exp = Expense.objects.get(journeyplan =
journey_plan, level = 0, expensegroup__id = 0)
            exp.expansessum = exp.expansessum +
form.instance.expansessum - car.expansessum
            exp.payedsum = exp.expansessum +
form.instance.payedsum - car.payedsum
            exp.save()
            journey_plan.expansessum =
journey_plan.expansessum + form.instance.expansessum -
car.expansessum
            journey_plan.payedsum = journey_plan.payedsum +
form.instance.payedsum - car.payedsum
            journey_plan.save()
            return super().form_valid(form)

class AccommodationCreateView(LoginRequiredMixin,
CreateView):
    model = Accommodation

    def get_success_url(self):
        return reverse('journey-plan',
kwargs={'journeyplan_id':
self.object.journeyplanpoint.journeyplan.id})

    def form_valid(self, form):
        if form.instance.added:
            point = form.instance.journeyplanpoint
            journey_plan = point.journeyplan
            exp = Expense.objects.get(journeyplan =
journey_plan, level = 0, expensegroup__id = 1)

```

```

        exp.expansessum = exp.expansessum +
form.instance.expansessum
        exp.payedsum = exp.payedsum +
form.instance.payedsum
        exp.save()
        journey_plan.expansessum =
journey_plan.expansessum + form.instance.expansessum
        journey_plan.payedsum = journey_plan.payedsum +
form.instance.payedsum
        journey_plan.save()
    return super().form_valid(form)

```

```

class AccommodationUpdateView(LoginRequiredMixin,
UpdateView):
    model = Accommodation

    def get_success_url(self):
        return reverse('journey-plan',
kwargs={'journeyplan_id':
self.object.journeyplanpoint.journeyplan.id})

    def form_valid(self, form):
        if form.instance.added:
            point = form.instance.journeyplanpoint
            journey_plan = point.journeyplan
            accommod = Accommodation.objects.get(id =
self.object.id)
            exp = Expense.objects.get(journeyplan =
journey_plan, level = 0, expensegroup__id = 1)
            exp.expansessum = exp.expansessum +
form.instance.expansessum - accommod.expansessum
            exp.payedsum = exp.expansessum +
form.instance.payedsum - accommod.payedsum
            exp.save()
            journey_plan.expansessum =
journey_plan.expansessum + form.instance.expansessum -
accommod.expansessum
            journey_plan.payedsum = journey_plan.payedsum +
form.instance.payedsum - accommod.payedsum
            journey_plan.save()
        return super().form_valid(form)

```

```

class JourneyPointCreateView(LoginRequiredMixin,
CreateView):
    model = JourneyPoint
    fields = ['journey',

```

```

        'point',
        'agreed']

    def get_success_url(self):
        return reverse('journey', kwargs={'journey_id':
self.object.journey.id})

    def form_valid(self, form):
        user = self.request.user
        traveler = Traveler.objects.get(user = user)
        journey = form.instance.journey
        if user.is_active and traveler and
journey.createdby == traveler:
            form.instance.byplanner = False
        elif user.is_active:
            planner = Planner.objects.get(user = user)
            requests =
PlannerRequest.objects.filter(planner = planner, journey =
journey, submitted = 1)
            if planner and requests:
                form.instance.byplanner = True
                form.instance.agreed = False
            return super().form_valid(form)

class JourneyPointAgreeView(LoginRequiredMixin,
UpdateView):
    model = JourneyPoint

    def get_success_url(self):
        return reverse('journey', kwargs={'journey_id':
self.object.journey.id})

    def form_valid(self, form):
        user = self.request.user
        traveler = Traveler.objects.get(user = user)
        journey = form.instance.journey
        if traveler and journey.createdby == traveler:
            form.instance.agreed = True
        return super().form_valid(form)

class JourneyPointDeleteView(DeleteView):
    model = JourneyPoint

    def get_success_url(self):
        return reverse('journey', kwargs={'journey_id':
self.object.journey.id})

```

```

class JourneyplanPointCreateView(LoginRequiredMixin,
CreateView):
    model = JourneyplanPoint
    fields = ['journeypoint',
              'journeyplan',
              'startdate',
              'enddate',
              'agreed']

    def get_success_url(self):
        return reverse('journey-plan',
kwargs={'journeyplan_id': self.object.journeyplan.id})

    def form_valid(self, form):
        user = self.request.user
        traveler = Traveler.objects.get(user = user)
        journey =
form.instance.journeyplan.plannerrequest.journey
        if user.is_active and traveler and
journey.createdby == traveler:
            form.instance.agreed = True
        elif user.is_active:
            planner = Planner.objects.get(user = user)
            requests =
PlannerRequest.objects.filter(planner = planner, journey =
journey, submitted = 1)
            if planner and requests:
                form.instance.agreed = False
        return super().form_valid(form)

class JourneyplanPointAgreeView(LoginRequiredMixin,
UpdateView):
    model = JourneyplanPoint

    def get_success_url(self):
        return reverse('journey-plan',
kwargs={'journeyplan_id': self.object.journeyplan.id})

    def form_valid(self, form):
        user = self.request.user
        traveler = Traveler.objects.get(user = user)
        journey =
form.instance.journeyplan.plannerrequest.journey
        if user.is_active and traveler and
journey.createdby == traveler:
            form.instance.agreed = True

```

```

        return super().form_valid(form)

class JourneyplanPointDeleteView(DeleteView):
    model = JourneyplanPoint

    def get_success_url(self):
        return reverse('journey-plan',
kwargs={'journeyplan_id': self.object.journeyplan.id})

class JourneyPlansView(ListView):
    model = JourneyPlan
    template_name = 'journeyplans.html'

    def get_queryset(self):
        journey_id = self.request.GET.get('journey')
        journey = Journey.objects.get(id = journey_id)
        user = self.request.user
        planner = Planner.objects.filter(user = user)
        allowed = False
        if user.is_active and planner:
            requests =
PlannerRequest.objects.filter(journey = journey, planner =
planner, submitted = 1)
            if requests:
                allowed = True
            elif user.is_active:
                traveler = Traveler.objects.filter(user = user)
                if traveler:
                    if journey.createdby == traveler or
traveler in journey.shares:
                        allowed = True
            if allowed:
                context_point ["journey"] = journey
                plans =
JourneyPlan.objects.filter(plannerrequest__ journey = journey,
submitted = 1).prefetch_related('discussions').order_by('-
createdat')
                context_point ["plans"] = plans
        return context_point

class JourneyPlanView(DetailView):
    model = JourneyPlan

    def get_context_data(self, **kwargs):
        context_point = super().get_context_data(**kwargs)

```

```

        main_plan = JourneyPlan.objects.get(id =
self.kwargs['pk']).prefetch_related('discussions')
        journey = main_plan.plannerrequest.journey
        user = self.request.user
        planner = Planner.objects.filter(user = user)
        allowed = False
        if user.is_active and planner:
            requests =
PlannerRequest.objects.filter(journey = journey, planner =
planner, submitted = 1)
            if requests:
                allowed = True
        elif user.is_active:
            traveler = Traveler.objects.filter(user = user)
            if traveler:
                if journey.createdby == traveler or
traveler in journey.shares:
                    allowed = True
        if allowed:
            context_point ["journeyplan"] = main_plan
            cars = Car.objects.filter(journeyplan =
main_plan).order_by('departureat').prefetch_related('discussions
')
            context_point ["cars"] = cars
            transport =
RouteSectionPublictransport.objects.filter(routesection__startjo
urneyplanpoint__journeyplan =
main_plan).prefetch_related('discussions').order_by('departureat
')
            context_point ["transport"] = transport
            accommodation =
Accommodation.objects.filter(journeyplanpoint__journeyplan =
main_plan).prefetch_related('discussions').order_by('departureat
')
            context_point ["accommodation"] = accommodation
            routes =
RouteSection.objects.filter(startjourneyplanpoint__journeyplan =
main_plan).prefetch_related('discussions').order_by('startdate')
            context_point ["routes"] = routes
            requests =
PlannerRequest.objects.filter(journey = journey).order_by('-
createdat')
            context_point ["requests"] = requests
            expenses = Expense.objects.filter(journeyplan =
main_plan).prefetch_related('discussions').order_by('id')
            context_point ["expenses"] = expenses

```

```

        points =
JourneyplanPoint.objects.filter(journeyplan =
main_plan).order_by('startdate')
        context_point ["points"] = points
        return context_point

def discussions(request):
    user = request.user
    traveler = Traveler.objects.get(user = user)
    if traveler and user.is_active:
        planners_direct =
PlannerRequest.objects.filter(journey__createdby =
traveler).distinct('planner')
        planners_direct_list = []
        for planner in planners_direct:
            discussions = Discussion.objects.filter(Q(user1
= traveler.user, user2 = planner.user) | Q(user1 = planner.user,
user2 = traveler.user))
            if discussions:
                dis = {}
                dis['planner'] = planner
                dis['discussions'] = discussions
                planners_direct_list.append(dis)
        journeys = Journey.objects.filter(shares =
traveler)
        planners_shared = Planner.objects.filter(shares__in
= journeys)
        planners_shared_list = []
        for planner in planners_shared:
            if planner not in planners_direct_list:
                discussions =
Discussion.objects.filter(Q(user1 = traveler.user, user2 =
planner.user, private = False) | Q(user1 = planner.user, user2 =
traveler.user, private = False))
                if discussions:
                    dis = {}
                    dis['planner'] = planner
                    dis['discussions'] = discussions
                    planners_shared_list.append(dis)
        travelers_shared =
Traveler.objects.filter(shares__in = journeys)
        travelers_shared_list = []
        for traveler_shared in travelers_shared:
            discussions = Discussion.objects.filter(Q(user1
= traveler.user, user2 = traveler_shared.user) | Q(user1 =
planner.user, user2 = traveler.user))

```

```

        if discussions:
            dis = {}
            dis['traveler'] = traveler_shared
            dis['discussions'] = discussions
            travelers_shared_list.append(dis)
    if request.method == "POST":
        discussion_form = DiscussionForm(request.POST)
        if discussion_form.is_valid():
            discussion =
discussion_form.save(commit=False)
            discussion.user1 = user
            discussion.save()
            redirect("discussions")
    else:
        discussion_form = DiscussionForm()
        return render(request, "tpss/review.html", {"form":
discussion_form, "direct": planners_direct_list, "shared":
planners_shared_list, "travelers": travelers_shared_list })

class TravelerFeedbackCreateView(LoginRequiredMixin,
CreateView):
    model = Feedback
    fields = ['planner',
              'mark',
              'text']

    def get_success_url(self):
        return reverse('planner-feedbacks')

    def form_valid(self, form):
        user = self.request.user
        traveler = Traveler.objects.get(user = user)
        if user.is_active and traveler:
            form.instance.way = False
            feedbacks = Feedback.objects.filter(planner =
form.instance.planner, way = False).aggregate(Sum('mark'))
            number = feedbacks / feedbacks.count()
            planner.mark = number
            planner.save()
        return super().form_valid(form)

class PlannerFeedbackCreateView(LoginRequiredMixin,
CreateView):
    model = Feedback
    fields = ['traveler',
              'mark',

```

```

        'text']

    def get_success_url(self):
        return reverse('traveler-feedbacks')

    def form_valid(self, form):
        user = self.request.user
        planner = Planner.objects.get(user = user)
        if user.is_active and planner:
            form.instance.way = True
            feedbacks = Feedback.objects.filter(traveler =
form.instance.traveler, way = True).aggregate(Sum('mark'))
            number = feedbacks / feedbacks.count()
            traveler.mark = number
            traveler.save()
        return super().form_valid(form)

class JourneyPlanCreateView(LoginRequiredMixin,
CreateView):
    model = JourneyPlan
    fields = ['caption',
              'details',
              'expansessum',
              'payedsum',
              'chosen',
              'plannerrequest']

    def get_success_url(self):
        return reverse('journey-plan',
kwargs={'journeyplan_id': self.object.id})

    def form_valid(self, form):
        user = self.request.user
        planner = Planner.objects.get(user = user)
        if user.is_active and planner:
            form.instance.chosen = False
        elif user.is_active:
            traveler = Traveler.objects.get(user = user)
            if traveler:
                journey_id =
self.request.GET.get('journey')
                journey = Journey.objects.get(id =
journey_id, None)
                if not journey:

```

```

        journey =
Journey.objects.filter(createdby = traveler).order_by('-
createdat')[0]
        request = PlannerRequest(journey = journey,
agreed = True)
        request.save()
        form.instance.plannerrequest = request
        return super().form_valid(form)

class JourneyPlanUpdateView(LoginRequiredMixin,
UpdateView):
    model = JourneyPlan
    fields = ['caption',
              'details',
              'expansessum',
              'payedsum',
              'chosen']

    def get_success_url(self):
        return reverse('journey-plan',
kwargs={'journeyplan_id':
self.object.journeyplanpoint.journeyplan.id})

class PlannerRequestCancelView(LoginRequiredMixin,
UpdateView):
    model = PlannerRequest

    def get_success_url(self):
        return reverse('journey-request',
kwargs={'journey_id': self.object.journey.id})

    def form_valid(self, form):
        user = self.request.user
        planner = Planner.objects.get(user = user)
        request = PlannerRequest.objects.get(id =
self.object.id)
        if planner and planner == request.planner:
            form.instance.agreed = False
        elif user.is_active:
            traveler = Traveler.objects.get(user = user)
            if traveler and request.journey.createdby ==
traveler:
                form.instance.agreed = False
        return super().form_valid(form)

```

```

class RouteSectionCreateView(LoginRequiredMixin,
CreateView):
    model = RouteSection

    def get_success_url(self):
        return reverse('journey', kwargs={'journey_id':
self.object.startjourneyplanpoint.plannerrequest.journey.id})

class RouteSectionUpdateView(LoginRequiredMixin,
UpdateView):
    model = RouteSection

    def get_success_url(self):
        return reverse('journey', kwargs={'journey_id':
self.object.startjourneyplanpoint.plannerrequest.journey.id})

class RouteSectionDeleteView(LoginRequiredMixin,
DeleteView):
    model = RouteSection

    def get_success_url(self):
        return reverse('journey', kwargs={'journey_id':
self.object.startjourneyplanpoint.plannerrequest.journey.id})

class ExpenseCreateView(LoginRequiredMixin, CreateView):
    model = Expense

    def get_success_url(self):
        return reverse('journey', kwargs={'journey_id':
self.object.journeyplan.plannerrequest.journey.id})

    def form_valid(self, form):
        if form.level != 0:
            group = form.instance.expensegroup
            exp = Expense.objects.get(level = 0,
expensegroup = group)
            exp.expansessum = exp.expansessum +
form.instance.expansessum
            exp.payedsum = exp.expansessum +
form.instance.payedsum
            exp.save()
            return super().form_valid(form)

class ExpenseUpdateView(LoginRequiredMixin, UpdateView):
    model = Expense

```

```

        def get_success_url(self):
            return reverse('journey', kwargs={'journey_id':
self.object.journeyplan.plannerrequest.journey.id})

        def form_valid(self, form):
            if form.level != 0:
                group = form.instance.expensegroup
                exp_old = Expense.objects.get(id =
self.object.id)
                exp = Expense.objects.get(level = 0,
expensegroup = group)
                exp.expansessum = exp.expansessum +
exp_old.expansessum
                exp.payedsum = exp.expansessum +
exp_old.payedsum
                exp.save()
            return super().form_valid(form)

class ExpenseDeleteView(LoginRequiredMixin, DeleteView):
    model = Expense

    def get_success_url(self):
        return reverse('journey', kwargs={'journey_id':
self.object.journeyplan.plannerrequest.journey.id})

class ExpenseJourneyplanPointCreateView(LoginRequiredMixin,
CreateView):
    model = ExpenseJourneyplanPoint

    def get_success_url(self):
        return reverse('journey', kwargs={'journey_id':
self.object.journeyplan.plannerrequest.journey.id})

    def form_valid(self, form):
        group = form.instance.expensegroup
        exp = Expense.objects.get(level = 0, expensegroup =
group)
        exp.expansessum = exp.expansessum +
form.instance.expansessum
        exp.payedsum = exp.expansessum +
form.instance.payedsum
        exp.save()
        journey_plan =
form.instance.journeyplanpoint.journeyplan
        journey_plan.expansessum = journey_plan.expansessum
+ form.instance.expansessum

```

```

        journey_plan.payedsum = journey_plan.payedsum +
form.instance.payedsum
        journey_plan.save()
        return super().form_valid(form)

class ExpenseJourneyplanPointUpdateView(LoginRequiredMixin,
UpdateView):
    model = ExpenseJourneyplanPoint

    def get_success_url(self):
        return reverse('journey', kwargs={'journey_id':
self.object.journeyplan.plannerrequest.journey.id})

    def form_valid(self, form):
        group = form.instance.expensgroup
        exp_old = ExpenseJourneyplanPoint.objects.get(id =
self.object.id)
        exp = Expense.objects.get(level = 0, expensgroup =
group)
        exp.expansessum = exp.expansessum +
form.instance.expansessum - exp_old.expansessum
        exp.payedsum = exp.expansessum +
form.instance.payedsum - exp_old.payedsum
        exp.save()
        journey_plan =
form.instance.journeyplanpoint.journeyplan
        journey_plan.expansessum = journey_plan.expansessum
+ form.instance.expansessum - exp_old.expansessum
        journey_plan.payedsum = journey_plan.payedsum +
form.instance.payedsum - exp_old.payedsum
        journey_plan.save()
        return super().form_valid(form)

class ExpenseJourneyplanPointDeleteView(LoginRequiredMixin,
DeleteView):
    model = ExpenseJourneyplanPoint

    def get_success_url(self):
        return reverse('journey', kwargs={'journey_id':
self.object.journeyplan.plannerrequest.journey.id})

class JourneyPlanExpensesView(DetailView):
    model = JourneyPlan

    def get_context_data(self, **kwargs):
        context_point = super().get_context_data(**kwargs)

```

```

        main_plan = JourneyPlan.objects.get(id =
self.kwargs['pk']).prefetch_related('discussions')
        journey = main_plan.plannerrequest.journey
        user = self.request.user
        planner = Planner.objects.filter(user = user)
        allowed = False
        if user.is_active and planner:
            requests =
PlannerRequest.objects.filter(journey = journey, planner =
planner, submitted = 1)
            if requests:
                allowed = True
        elif user.is_active:
            traveler = Traveler.objects.filter(user = user)
            if traveler:
                if journey.createdby == traveler or
traveler in journey.shares:
                    allowed = True
        if allowed:
            added_expenses = []
            context_point ["journeyplan"] = main_plan
            expenses = Expense.objects.filter(journeyplan =
main_plan, level =
0).prefetch_related('discussions').order_by('id')
            for exp in expenses:
                exp.related_expenses =
Expense.objects.filter(journeyplan = main_plan, expensegroup =
exp.expensegroup, ~Q(level =
0)).prefetch_related('discussions').order_by('id')
                exp.points =
ExpenseJourneyplanPoint.objects.filter(expense =
exp).order_by('id')
                exp.transport_expenses =
RoutesectionPublictransport.objects.filter(routesection__startjo
urneyplanpoint__journeyplan = main_plan, added =
False).order_by('departureat').values_list("routesection",
"departureat", "departurepoint", "arriveat", "arrivepoint",
"transportcompany", "note", "expansessum", "payedsun")
                exp.accommodation_expenses =
Accommodation.objects.filter(journeyplanpoint__journeyplan =
main_plan, added =
False).order_by('checkin').values_list("journeyplanpoint",
"checkin", "checkout", "service", "expansessum", "payedsun")
                exp.car_expenses =
Car.objects.filter(journeyplan = main_plan, added =
False).order_by('departureat').values_list("rentalcompany",

```

```

"mark", "model", "departureat", "arriveat", "departurepoint",
"arrivepoint", "expansessum", "payedsum")
        if exp.expensegroup not in added_expenses:
            added_expenses.append(exp.expensegroup)
        context_point ["expenses"] = expenses
        add_expenses =
Expensegroup.objects.filter(~Q(id__in = added_expenses))
        context_point ["add_expenses"] = add_expenses
        return context_point

class JourneyPlanAccommodationsView(DetailView):
    model = JourneyPlan

    def get_context_data(self, **kwargs):
        context_point = super().get_context_data(**kwargs)
        main_plan = JourneyPlan.objects.get(id =
self.kwargs['pk']).prefetch_related('discussions')
        journey = main_plan.plannerrequest.journey
        user = self.request.user
        planner = Planner.objects.filter(user = user)
        allowed = False
        if user.is_active and planner:
            requests =
PlannerRequest.objects.filter(journey = journey, planner =
planner, submitted = 1)
            if requests:
                allowed = True
            elif user.is_active:
                traveler = Traveler.objects.filter(user = user)
                if traveler:
                    if journey.createdby == traveler or
traveler in journey.shares:
                        allowed = True
            if allowed:
                added_expenses = []
                context_point ["journeyplan"] = main_plan
                accommodations =
Accommodation.objects.filter(journeyplanpoint__journeyplan =
main_plan).prefetch_related('discussions').order_by('departureat
')
                context_point ["accommodations"] =
accommodations
                points =
JourneyplanPoint.objects.filter(journeyplan = main_plan,
journeyplanpoint__point__pointtype = 0, accommodations__in =
accommodations).order_by('startdate')

```

```

        start = journey.startdate
        start_s = datetime.datetime(start.year,
start.month, start.day, 0, 0)
        end = journey.enddate
        start_e = datetime.datetime(start.year,
start.month, start.day, 23, 59)
        cities = []
        while start_e < end:
            accommodations =
Accommodation.objects.filter(journeyplanpoint__journeyplan =
main_plan, checkin__gte = start_s, checkin__lte = start_e)
            if not accommodations:
                points =
JourneyplanPoint.objects.filter(journeyplan = main_plan)
                points = points.filter(Q(startdate__lt
= start_s, enddate__lt = start_e, enddate__gt = start_s) |
Q(startdate__gt = start_s, startdate__lt = start_e, enddate__gt
= start_e) | Q(startdate__lt = start_s, enddate__gt = start_e))
                if points:
                    cities.append(points)
                else:
                    cities.append('empty')
            else:
                cities.append(accommodations)
            start = start + datetime.timedelta(days=1)
            start_s = datetime.datetime(start.year,
start.month, start.day, 0, 0)
            start_e = datetime.datetime(start.year,
start.month, start.day, 23, 59)
            context_point ["cities"] = cities
        return context_point

```

```

class JourneyPlanRouteView(DetailView):
    model = JourneyPlan

    def get_context_data(self, **kwargs):
        context_point = super().get_context_data(**kwargs)
        main_plan = JourneyPlan.objects.get(id =
self.kwargs['pk']).prefetch_related('discussions')
        journey = main_plan.plannerrequest.journey
        user = self.request.user
        planner = Planner.objects.filter(user = user)
        allowed = False
        if user.is_active and planner:

```

```

        requests =
PlannerRequest.objects.filter(journey = journey, planner =
planner, submitted = 1)
        if requests:
            allowed = True
        elif user.is_active:
            traveler = Traveler.objects.filter(user = user)
            if traveler:
                if journey.createdby == traveler or
traveler in journey.shares:
                    allowed = True
        if allowed:
            added_expenses = []
            context_point ["journeyplan"] = main_plan
            sections =
Routesection.objects.filter(startjourneyplanpoint__journeyplan =
main_plan).order_by('startjourneyplanpoint')
            context_point ["sections"] = sections
            section_prev = None
            section_list = []
            for section in sections:
                if section_prev:
                    if
section_prev.endjourneyplanpoint.enddate !=
section.startjourneyplanpoint.startdate:
                        info = {}
                        info["period"] =
(section_prev.endjourneyplanpoint.enddate,
section.startjourneyplanpoint.startdate)
                        info["points"] =
JourneyplanPoint.objects.filter(journeyplan = main_plan,
startdate__gte = section_prev.endjourneyplanpoint.enddate,
enddate__lte =
section.startjourneyplanpoint.startdate).order_by('startdate')
                    else:
                        info = {}
                        info["period"] =
(section.startjourneyplanpoint.startdate,
section.endjourneyplanpoint.enddate)
                        info["section"] = section
                else:
                    info = {}
                    info["period"] =
(section.startjourneyplanpoint.startdate,
section.endjourneyplanpoint.enddate)
                    info["section"] = section

```

```
        section_list.append(section)
        section_prev = section
context_point ["sections_details"] = section_list
return context_point
```

ДОДАТОК В
Слайди презентації

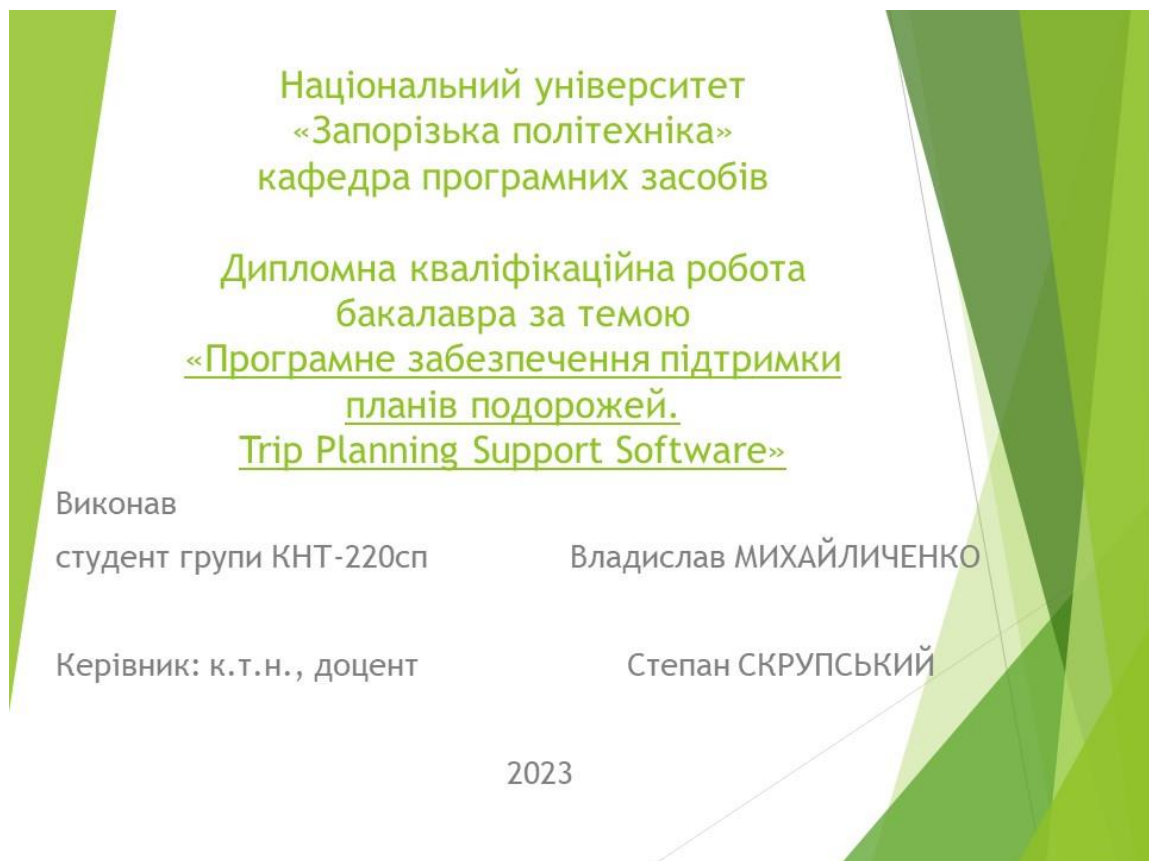


Рисунок В.1 – Слайд титульний

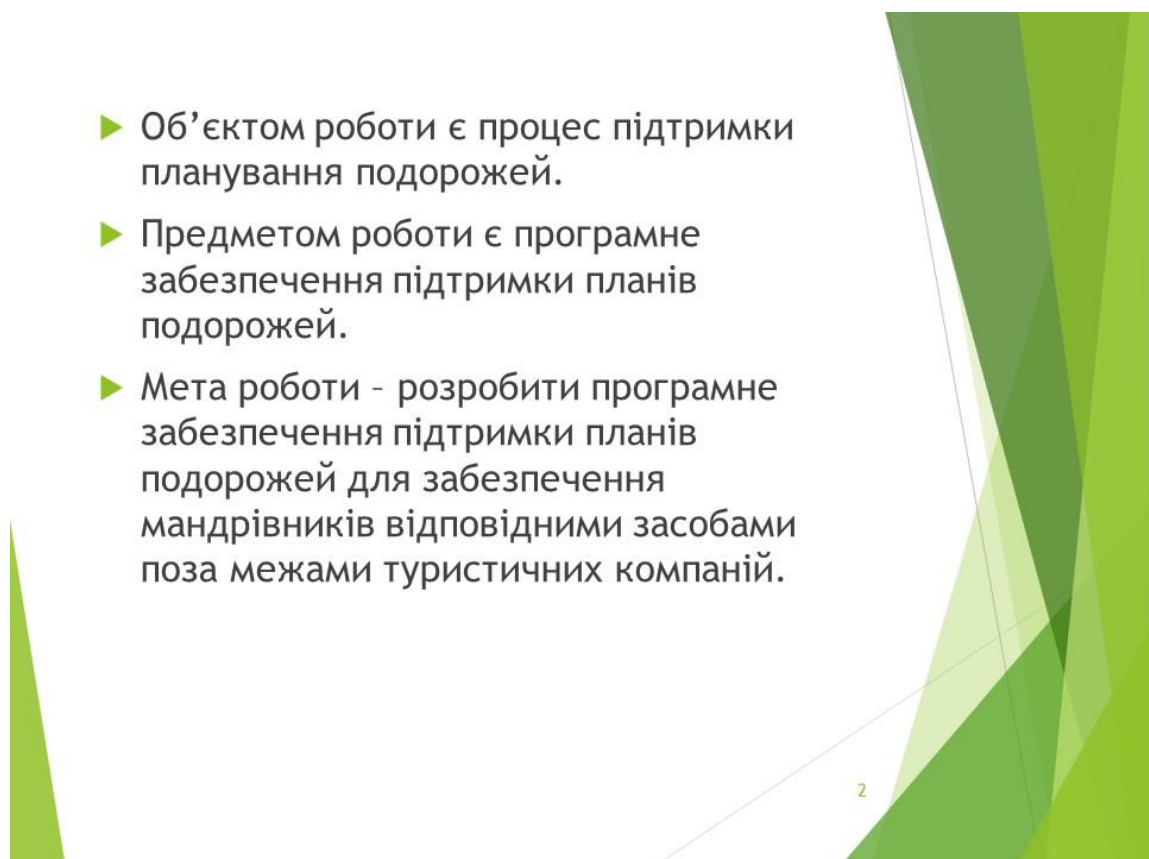


Рисунок В.2 – Мета роботи

Завдання роботи

- аналіз програмних засобів підтримки планів подорожей і визначення концепції програми;
- проєктування програмного забезпечення підтримки планів подорожей;
- реалізація вебдодатку підтримки планів подорожей.

Рисунок В.3 – Завдання роботи

Результати аналізу програмних засобів підтримки планів подорожей

Ознака	Triptile	Wanderlog	Goasklocal	Global Game Plan	Розробка
Створення плану подорожі	Так	Так	Ні	Ні	Так
Формування бюджету подорожі	Так	Так	Ні	Ні	Так
Пошук експертів	Ні	Ні	Так	Так	Так
Реєстрація експертів	Ні	Ні	Ні	Ні	Так
Розділення роботи з подорожжю з мандрівниками	Ні	Ні	Ні	Ні	Так

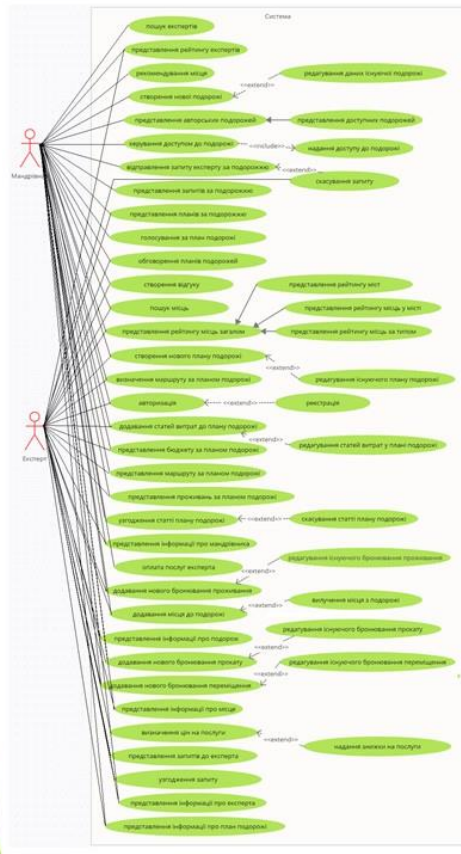
Рисунок В.4 – Результати аналізу програмних засобів підтримки планів подорожей

Порівняння мов Python і PHP для розробки програми

Ознака	Python	PHP
Забезпечення потужного вебфреймворку	Так	Так
Спрощення процесу розробки	Вище	Менше
Керування вразливостями	Краще	Гірше
Можливості розширення	Перспективніші за рахунок простішої роботи з засобами штучного інтелекту	Менш перспективні в даному розрізі

5

Рисунок В.5 – Порівняння мов Python і PHP для розробки програми



Діаграма прецедентів

6

Рисунок В.6 – Діаграма прецедентів

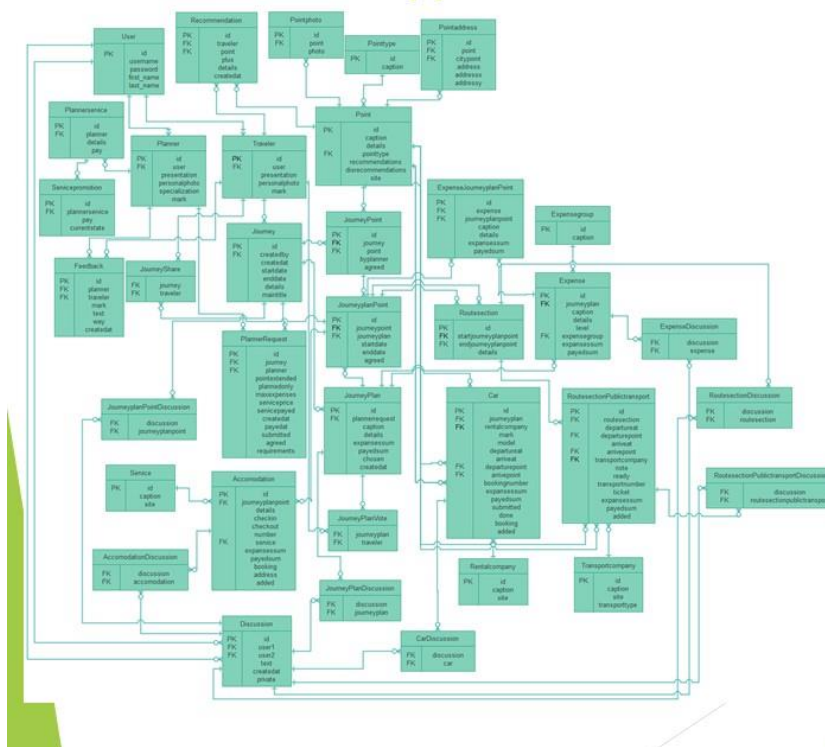


Рисунок В.7 – Схема БД

Діаграма класів спільних для експерта та мандрівника

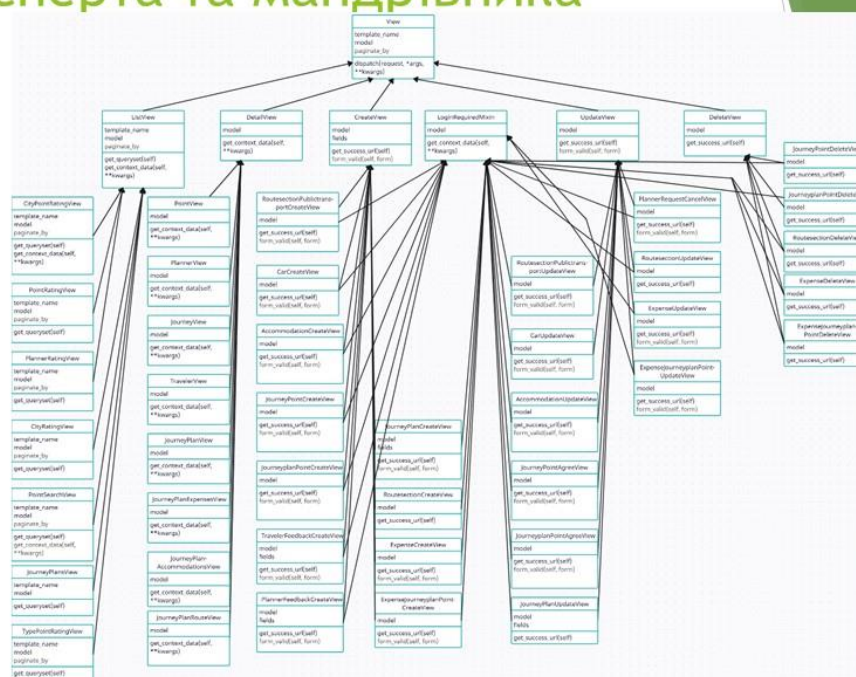


Рисунок В.8 – Діаграма класів спільних для експерта та мандрівника

Засоби мандрівника в програмі

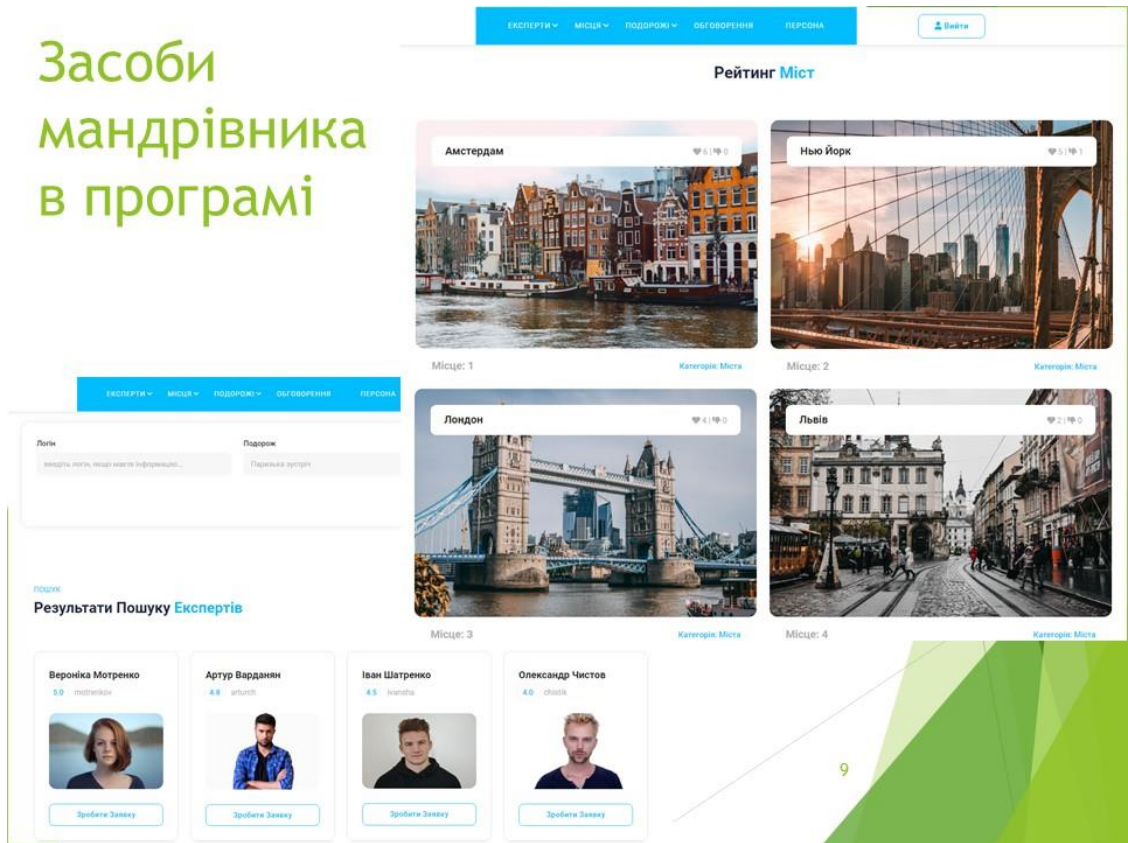


Рисунок В.9 – Засоби мандрівника в програмі

Перегляд інформації про подорож

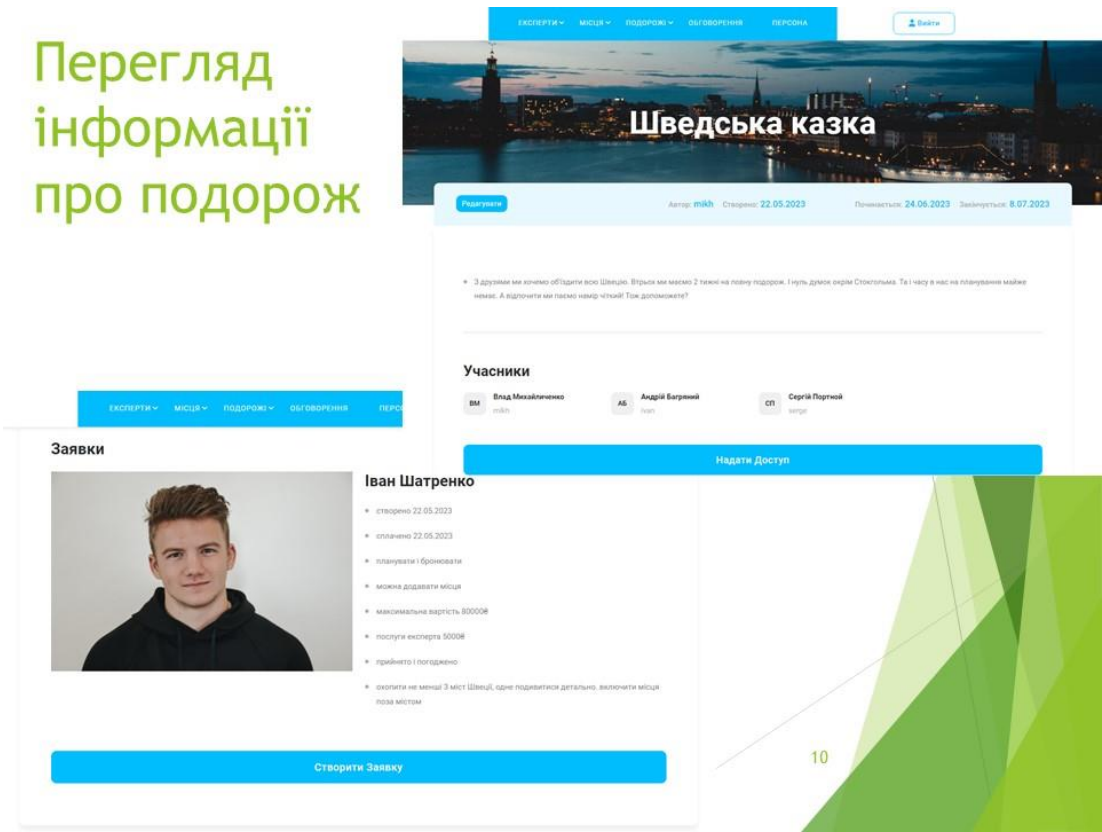


Рисунок В.10 – Перегляд інформації про подорож

Планування подорожі експертом

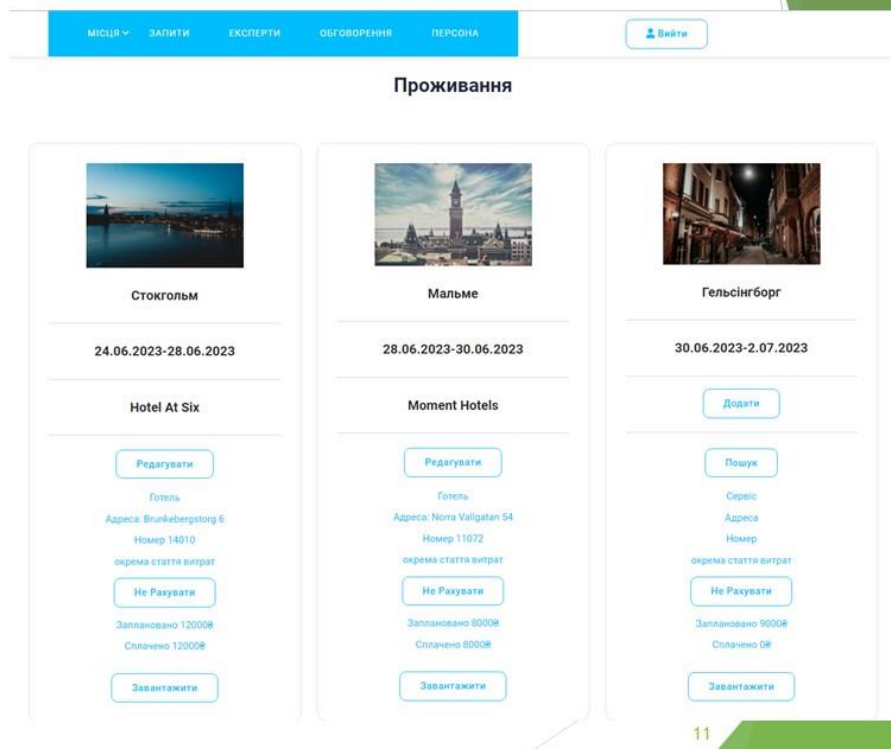


Рисунок В.11 – Планування подорожі експертом

Висновки за роботою

- вебсистему було створено, застосовуючи мову Python з фреймворком Django і системою MySQL для роботи з базою даних, при моделюванні сценаріїв роботи мандрівника та експерта з системою, створенні структури бази даних визначено структуру шарів для роботи з планами подорожей;
- програмне забезпечення дозволяє знаходити експертів для планування подорожей, які можуть створювати план або плани подорожі, готувати всі необхідні бронювання для мандрівника, також мандрівники можуть знаходити нові місця, додавати їх до структури подорожі, визначаючи таким чином вимоги до планування.

Рисунок В.12 – Висновки за роботою