МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ТИХОМИРОВ ГЕРМАН ОЛЕКСАНДРОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
д-р техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

ПРОЄКТУВАННЯ БАЗИ ДАНИХ ВНУТРІШНЬОГО ОБЛІКУ ТУРИСТИЧНОГО АГЕНТСТВА

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Петро НІКОЛЮК, проф. кафедри
інформаційних технологій,
д-р фіз-мат. наук, професор
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

Вінниця–2026

АНОТАЦІЯ

Спеціальність 122 «Комп’ютерні науки», освітня програма «Комп’ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2024. Кваліфікаційна робота присвячена проєктуванню бази даних внутрішнього обліку туристичного агентства. Розроблено модель даних для клієнтів, працівників, напрямів, готелів, турів, бронювань, платежів, документів і заявок із сайту. Практичну реалізацію виконано засобами Django та PostgreSQL; створено користувацький веб-інтерфейс і адміністративну панель для опрацювання бронювань.

Ключові слова: туристичне агентство, база даних, Django, PostgreSQL, бронювання, внутрішній облік, веб-застосунок.

ABSTRACT

Tykhomyrov H. Development of a web application for monitoring local educational institutions. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2024. The qualification work is devoted to designing a database for internal accounting in a travel agency. The data model covers clients, employees, destinations, hotels, tours, bookings, payments, travel documents and website requests. The practical implementation uses Django and PostgreSQL and includes a client web interface and an administrative panel for booking management.

Keywords: travel agency, database, Django, PostgreSQL, booking, internal accounting, web application.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Таблиця 0.1 – Перелік умовних позначень

Позначення	Пояснення
БД	база даних
СУБД	система управління базами даних
ORM	об'єктно-реляційне відображення, що пов'язує класи програми з таблицями бази даних
CRUD	операції створення, читання, оновлення і видалення записів
ER-модель	модель «сутність-зв'язок», що показує структуру предметної області
UI	користувацький інтерфейс
CSRF	тип веб-атаки та механізм захисту форм від підробки міжсайтових запитів
URL	уніфікований покажчик ресурсу
SQL	мова структурованих запитів

ВСТУП

Актуальність теми дослідження зумовлена переходом туристичного бізнесу до цифрових моделей обслуговування, у яких швидкість опрацювання заявки й точність внутрішнього обліку безпосередньо впливають на якість сервісу. Туристичне агентство працює не лише з переліком турів, а й з персональними даними клієнтів, історією бронювань, платежами, туристичними документами, статусами звернень та діями працівників. Якщо ці дані зберігаються у розрізнених таблицях або паперових формах, виникають дублювання, затримки в пошуку інформації та складність контролю відповідальних осіб [1-3].

Для малого й середнього туристичного агентства особливо важливо мати систему, яка не потребує надмірних витрат на впровадження, але забезпечує централізовану базу даних, зрозумілий інтерфейс для працівників і доступний веб-канал для клієнтів. Саме тому в роботі розглядається проєктування бази даних внутрішнього обліку з подальшою реалізацією у вигляді Django-застосунку. Обраний підхід поєднує нормалізовану модель даних, готову адміністративну панель, автентифікацію користувачів і можливість подальшого розширення функціоналу.

Метою роботи є проєктування та практична реалізація бази даних внутрішнього обліку туристичного агентства, що підтримує роботу з клієнтами, турами, напрямами, готелями, бронюваннями, платежами, документами й заявками з веб-сайту. Досягнення цієї мети передбачає не лише опис таблиць, а й перевірку того, як модель працює у реальному користувацькому сценарії: від перегляду туру до створення бронювання, його підтвердження менеджером і відображення статусу в особистому кабінеті клієнта.

Для досягнення поставленої мети визначено такі завдання: проаналізувати предметну область туристичного агентства; виділити основні бізнес-процеси внутрішнього обліку; порівняти існуючі підходи й програмні рішення; сформулювати функціональні та нефункціональні вимоги; побудувати

концептуальну, логічну й фізичну моделі бази даних; реалізувати моделі засобами Django ORM і PostgreSQL; налаштувати адміністративний та користувацький інтерфейси; виконати функціональне тестування основних сценаріїв.

Об'єктом дослідження є процес внутрішнього обліку туристичного агентства. Предметом дослідження є структура бази даних і програмні механізми, що забезпечують облік клієнтів, турів, бронювань, оплат, документів та заявок. Така постановка дозволяє розглядати систему не як ізольований веб-сайт, а як інформаційне середовище, у якому зовнішній інтерфейс клієнта пов'язаний із внутрішньою роботою менеджерів і адміністратора.

У роботі застосовано методи системного аналізу, порівняльного аналізу, моделювання даних, нормалізації відношень, об'єктно-реляційного відображення, функціонального тестування та аналізу користувацьких сценаріїв. Теоретичною основою стали підходи до проєктування баз даних, вимог до програмного забезпечення, якості програмних систем і безпечної веб-розробки [4-6], [7-8]. Практична частина спирається на офіційну документацію Django [9-13], PostgreSQL] [14-16], Python [17] , HTML [18] і CSS [19]

Практичне значення одержаних результатів полягає в отриманні працездатного навчально-прикладного програмного продукту. Система може бути використана як демонстраційний прототип для малого туристичного агентства або як база для подальшого розвитку: додавання звітів, автоматичного розрахунку залишку місць, розмежування ролей працівників, експорту документів і розширеної аналітики продажів.

Апробація результатів у межах публічних наукових заходів не проводилася. Практична перевірка виконана шляхом запуску веб-застосунку, наповнення тестовими даними, перегляду користувацьких сторінок, створення бронювання, перевірки особистого кабінету та роботи адміністративної панелі. Екранні форми, наведені у звіті, підтверджують відповідність реалізації заявленим функціональним вимогам.

Структурно звіт складається з анотацій українською та англійською мовами, змісту, переліку умовних позначень, вступу, трьох розділів, висновків, списку використаних посилань і додатків. Перший розділ присвячено аналізу предметної області та постановці задачі. У другому розділі описано проєктування бази даних. У третьому розділі розкрито реалізацію, інтерфейси, тестування та напрями оптимізації розробленої системи.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальна характеристика діяльності туристичних агентств

Туристичне агентство є організацією, що здійснює посередницьку діяльність між туристом і постачальниками туристичних послуг. У межах типової операційної діяльності агентство консультує клієнта, підбирає напрям і готель, перевіряє умови туру, фіксує бронювання, супроводжує оплату, формує комплект документів та контролює зворотний зв'язок після подорожі. Такий процес поєднує комерційну, сервісну, інформаційну й документальну складові.

На прикладі агентств, що працюють із масовими туристичними напрямами, можна виділити кілька стабільних груп даних: клієнти, працівники, напрями, готелі, тури, бронювання, платежі та документи. Частина цих даних має довідковий характер і змінюється нечасто, наприклад перелік країн, міст, готелів або працівників. Інша частина є операційною, оскільки створюється щоденно під час роботи з клієнтами.

Внутрішній облік туристичного агентства відрізняється від звичайного каталогу товарів тим, що одна послуга має часові межі, кількість місць, прив'язку до маршруту, готелю, менеджера й конкретних документів. Тур не можна розглядати як простий запис із назвою та ціною: він пов'язаний з датами, умовами харчування, доступністю, можливістю зміни статусу бронювання та фінансовим супроводом.

У роботі агентства важливу роль відіграють працівники різних ролей. Менеджер спілкується з клієнтом і супроводжує бронювання, бухгалтер контролює платежі, керівник або адміністратор аналізує загальний стан продажів та якість обліку. Навіть у невеликій команді необхідно бачити, хто відповідає за конкретну заявку, коли вона була створена, який має статус і чи були внесені платежі.

Клієнтський контур також має власну специфіку. Сучасний клієнт очікує можливості переглянути тури онлайн, залишити заявку або створити

бронювання без обов'язкового телефонного дзвінка. При цьому туристичне агентство не може повністю автоматизувати весь процес, адже остаточне підтвердження умов часто потребує участі менеджера. Тому доцільною є гібридна модель: клієнт ініціює дію на сайті, а менеджер опрацьовує її в адміністративному модулі.

З погляду бази даних предметна область є реляційною за своєю природою. Клієнт може мати багато бронювань, тур може входити до багатьох бронювань, бронювання може мати кілька платежів і документів. Такі зв'язки зручно описуються зовнішніми ключами та обмеженнями цілісності. Водночас потрібно уникати зберігання важливих даних лише у вільних текстових полях, оскільки це ускладнює пошук, фільтрацію та звітність.

Окремим аспектом є історичність даних. Після створення бронювання інформація про клієнта, тур, суму й менеджера стає частиною облікової історії. Видалення довідкового запису не повинно руйнувати минулі операції, тому для ключових сутностей доцільно застосовувати захисні правила видалення. У реалізованому проєкті ця ідея відображена через використання `on_delete=PROTECT` для зв'язків, що формують історію бронювання.

Таким чином, туристичне агентство як предметна область потребує системи, яка підтримує не лише зберігання довідників, а й цілісний життєвий цикл заявки. База даних має бути достатньо нормалізованою, щоб не допускати дублювання, але водночас зрозумілою для практичного використання в адміністративному інтерфейсі. Саме ця вимога визначає подальший вибір структури сутностей і засобів реалізації.

Додатково слід зазначити, що туристична послуга має високу залежність від часу. Зміна дати вильоту, готелю або ціни може суттєво вплинути на рішення клієнта, тому система обліку повинна швидко відображати актуальний стан пропозиції. У розробленій моделі дата початку й дата завершення туру є окремими полями, що дає змогу сортувати пропозиції, аналізувати сезонність і надалі будувати календарні звіти.

Комунікація з клієнтом часто починається ще до створення бронювання. Людина може лише цікавитися напрямом, порівнювати варіанти або уточнювати бюджет. Через це в моделі передбачено окрему сутність TourRequest. Вона не перевантажує таблицю бронювань попередніми зверненнями, але зберігає для менеджера контакт і побажання потенційного покупця.

Важливою рисою туристичного обліку є поєднання персональних, фінансових і сервісних даних. Наприклад, одне бронювання одночасно містить дані клієнта, маршрут, готель, кількість туристів, суму, оплату й документи. Це означає, що помилка в одному елементі може вплинути на весь процес. Тому модель даних повинна бути не лише повною, а й цілісною.

Таблиця 1.1 – Основні об'єкти внутрішнього обліку туристичного агентства

Об'єкт	Зміст даних	Роль у процесі
Клієнт	ПІБ, контакти, паспортні дані, примітки	Ідентифікує замовника туристичної послуги
Працівник	ПІБ, посада, контакти, ознака активності	Визначає відповідального менеджера або службову роль
Напрям	Країна, місто або курорт, опис	Служить довідником для класифікації турів
Готель	Назва, категорія, адреса, напрям	Деталізує умови розміщення у турі
Тур	Назва, дати, ціна, місця, харчування	Є товарною пропозицією агентства
Бронювання	Клієнт, тур, менеджер, кількість туристів, сума, статус	Фіксує основну операцію продажу
Платіж	Дата, сума, спосіб оплати	Підтверджує фінансовий супровід бронювання
Документ	Тип, номер, дата, файл	Забезпечує документальне оформлення туру
Заявка	Контакти, побажання, бажана країна, статус	Фіксує попередній інтерес клієнта

1.2. Аналіз бізнес-процесів внутрішнього обліку туристичного агентства

Перший бізнес-процес пов'язаний з обліком клієнтів. Для кожного клієнта необхідно зберігати ім'я, прізвище, телефон, електронну адресу, номер паспорта та службові примітки. У розробленій системі клієнт може бути пов'язаний з обліковим записом користувача Django через відношення один-до-одного. Це дозволяє поєднати внутрішній облік і доступ клієнта до особистого кабінету.

Другий бізнес-процес стосується обліку працівників. Працівник має ім'я, прізвище, посаду, контакти та ознаку активності. Для задачі бронювання важливо відрізняти активних менеджерів від інших працівників, оскільки саме менеджер автоматично призначається на нову заявку. Така логіка зменшує ручне навантаження на адміністратора і забезпечує первинний розподіл відповідальності.

Третій процес охоплює ведення довідника напрямів і готелів. Напрям описується країною, містом або курортом і текстовим описом. Готель пов'язується з напрямом і має назву, категорію за кількістю зірок та адресу. Винесення цих даних в окремі сутності дозволяє повторно використовувати їх у різних турах, не дублюючи країну, місто або назву готелю в кожному записі туру.

Четвертий процес - облік туристичних пропозицій. Тур містить назву, напрям, готель, дату початку, дату завершення, загальну кількість місць, базову ціну, тип харчування, посилання на зображення та статус активності. Поле активності дає змогу тимчасово приховувати тур із користувацького сайту без видалення запису з бази даних.

П'ятий процес - створення бронювання. Бронювання є центральною сутністю, яка пов'язує клієнта, тур і менеджера. У ньому фіксуються дата бронювання, кількість туристів, загальна сума, статус і коментар. Саме навколо бронювання формуються платежі, документи й управлінські дії в адміністративній панелі.

Шостий процес відповідає за фінансовий супровід. Платіж пов'язується з бронюванням і містить дату, суму, спосіб оплати та примітку. Така структура дозволяє підтримувати як повну, так і часткову оплату. У майбутньому на її

основі можна сформувати звіти про неоплачені бронювання, суму надходжень за період або ефективність менеджерів.

Сьомий процес пов'язаний з туристичними документами. До бронювання можуть бути прикріплені договір, ваучер, страховий поліс або квиток. Для кожного документа зберігаються тип, номер, дата видачі та файл. Це дає змогу не змішувати документи з текстом бронювання, а підтримувати структурований облік супровідних матеріалів.

Окремий процес становить обробка заявок із сайту. Гість, який не увійшов до системи, може залишити заявку на конкретний тур або на індивідуальний підбір. У такому записі зберігаються контактні дані, бажана країна, кількість туристів, коментар і статус опрацювання. Завдяки цьому агентство не втрачає потенційного клієнта навіть тоді, коли він ще не готовий реєструватися.

Узагальнення бізнес-процесів показує, що модель повинна підтримувати кілька рівнів роботи: довідковий, операційний, фінансовий, документальний і користувацький. Кожен рівень має власні сутності, але всі вони сходяться в бронюванні як ключовій операції внутрішнього обліку. Саме це обґрунтовує центральне місце таблиці Booking у подальшій моделі даних.

На практиці внутрішній облік має забезпечувати не лише зберігання даних, а й швидкий доступ до них. Менеджеру потрібні пошук за клієнтом, фільтр за статусом, перегляд платежів у межах бронювання та можливість змінити статус без переходу між багатьма екранами. Ці вимоги враховано в налаштуваннях Django Admin через `list_display`, `list_filter`, `search_fields`, `list_editable` та вбудовані inline-форми.

У процесі оформлення бронювання менеджер має бачити не ізольований запис, а повний контекст: хто клієнт, який тур обрано, скільки туристів їде, яка сума, чи були платежі та які документи вже видано. Саме тому платежі й документи розміщено як пов'язані записи бронювання, а в адміністративному інтерфейсі вони доступні без переходу до окремого розділу.

Бізнес-процес скасування також потребує акуратного обліку. Видалення бронювання призвело б до втрати історії, тому в системі використано зміну статусу. Такий підхід дозволяє не показувати скасоване бронювання клієнту в активному кабінеті, але залишати запис доступним для службового аналізу.

Заявки з сайту і бронювання мають різний юридичний та обліковий зміст. Заявка означає інтерес клієнта, а бронювання - вже оформлену операцію. Винесення цих понять у різні таблиці робить модель точнішою і дозволяє будувати окрему воронку: від заявки до підтвердженого туру.

Таблиця 1.2 – Бізнес-процеси та дані, які вони формують

Процес	Вхідні дані	Результат у системі	Відповідальна роль
Реєстрація клієнта	Логін, ПІБ, email, телефон, паспорт	User і Client	Клієнт
Підбір туру	Пошуковий запит, напрям	Список активних турів	Клієнт / менеджер
Створення заявки	Контакти, побажання, кількість туристів	TourRequest зі статусом «Нова»	Гість
Створення бронювання	Тур, кількість туристів, коментар	Booking зі статусом «Нове»	Клієнт
Підтвердження	Обране бронювання	Статус «Підтверджено»	Менеджер
Оплата	Дата, сума, спосіб оплати	Payment у межах бронювання	Бухгалтер
Оформлення документів	Тип і номер документа, файл	TravelDocument	Менеджер
Скасування	Поточне бронювання	Статус «Скасовано»	Клієнт / менеджер

1.3. Аналіз існуючих програмних рішень

Для автоматизації обліку туристичні агентства можуть використовувати електронні таблиці, універсальні CRM, галузеві CRM, бухгалтерські системи або власні веб-застосунки. Кожен варіант має переваги та обмеження. Електронні таблиці швидко впроваджуються, але погано контролюють зв'язки між даними. Універсальні CRM зручні для комунікацій, проте не завжди враховують специфіку туристичних документів і турів із датами.

Галузеві CRM для туризму зазвичай мають більше готових функцій, але можуть бути надмірними для навчального проєкту або малого агентства. Вони часто потребують оплати підписки, налаштування інтеграцій і адаптації до внутрішніх процесів конкретної організації. Для дипломної роботи важливішим є прозоре проєктування моделі та можливість пояснити кожне рішення, тому власна реалізація є обґрунтованою.

Локальні бухгалтерські системи добре підтримують фінансовий облік, однак не замінюють повноцінний облік турів, заявок і клієнтського веб-інтерфейсу. Якщо туристичне агентство веде каталог турів в одному місці, платежі в іншому, а заявки в месенджерах, виникає проблема фрагментації даних. У такій ситуації менеджер витрачає час не на обслуговування клієнта, а на звіряння інформації.

Власний веб-застосунок на Django дає змогу поєднати базу даних, адміністративний інтерфейс і публічні сторінки в одному проєкті. Django Admin особливо корисна для внутрішнього обліку, оскільки автоматично надає CRUD-операції, фільтри, пошук, редагування пов'язаних записів і групові дії. Це знижує обсяг коду, який потрібно писати вручну, але не скасовує необхідності якісного проєктування моделей [9, 12].

PostgreSQL обрано як основну СУБД через підтримку транзакцій, зовнішніх ключів, обмежень цілісності, індексів, розширених типів даних і стабільної роботи у веб-застосунках. Для задачі внутрішнього обліку важливими є точність зберігання сум, коректна робота з датами, надійність зв'язків між таблицями та можливість подальшої оптимізації запитів [14-16].

Альтернативою міг бути SQLite, який простіший для запуску, але менш придатний для реальної багатокористувацької експлуатації. MySQL також може бути використаний, однак у проєкті обрано PostgreSQL, оскільки він добре поєднується з Django, має розвинені можливості контролю даних і широко використовується у production-середовищах. У контексті дипломної роботи це дозволяє показати не лише прототип, а й наближення до практичної архітектури.

Порівняльний аналіз існуючих рішень свідчить, що для поставленої задачі найкращим є підхід, у якому власна база даних реалізує специфіку предметної області, а готові можливості фреймворку використовуються для типових технічних задач. Таке поєднання скорочує час розробки, підвищує якість структури та залишає можливість для майбутнього розвитку функціоналу.

Власна реалізація також дає перевагу прозорості. У навчальному проєкті можна простежити, як кожна вимога перетворюється на поле, зв'язок, форму або адміністративне налаштування. Це важливо для кваліфікаційної роботи, оскільки демонструє не тільки результат, а й логіку прийняття проєктних рішень.

Недоліком власної реалізації є потреба в супроводі. На відміну від готової CRM, розроблена система потребує налаштування середовища, оновлення залежностей, резервного копіювання й безпекової підтримки. Однак для навчального прототипу ці витрати є прийнятними, а отриманий результат краще показує навички проєктування бази даних.

Перевагою Django є наявність готових компонентів, які не заважають моделюванню предметної області. Розробник описує власні сутності, але отримує міграції, форми, автентифікацію і адміністративний інтерфейс. Це дає змогу зосередитися на внутрішньому обліку, а не на повторному створенні базових веб-механізмів.

Таблиця 1.3 – Порівняння підходів до автоматизації обліку

Підхід	**Переваги**	**Недоліки**	**Висновок для проєкту**
Електронні таблиці	Швидкий старт, знайомий інтерфейс, не потребує розробки	Слабкий контроль зв'язків, дублювання, складна спільна робота	Придатні лише для початкового обліку
Універсальна CRM	Контакти, задачі, комунікації, готові звіти	Не завжди враховує тури, документи й платежі	Корисна, але потребує адаптації
Галузева CRM	Ближча до туристичної специфіки, багато готових функцій	Підписка, залежність від постачальника, складність навчання	Надмірна для навчального прототипу

Підхід	Переваги	Недоліки	Висновок для проєкту
Бухгалтерська система	Сильний фінансовий облік, документи, звіти	Слабкий клієнтський веб-інтерфейс, не покриває каталог турів	Може бути інтеграцією, але не основою
Власний Django-застосунок	Повний контроль моделі, швидка адмін-панель, розширюваність	Потребує супроводу й налаштування середовища	Найкраще відповідає задачі дипломної роботи

1.4. Формулювання вимог до інформаційної системи

Функціональні вимоги до системи сформульовано на основі аналізу бізнес-процесів. Система повинна забезпечувати ведення клієнтів, працівників, напрямів, готелів, турів, бронювань, платежів, документів і заявок із сайту. Кожна з цих операцій має виконуватися через структуровані форми, а не через довільні текстові записи, оскільки саме структура забезпечує подальшу фільтрацію і звітність.

Для клієнтської частини визначено вимоги перегляду активних турів, пошуку за назвою, країною, містом або готелем, фільтрації за напрямом, відкриття сторінки конкретного туру, реєстрації, входу до системи, створення бронювання та перегляду власних бронювань. Гість, який не авторизований, повинен мати можливість залишити заявку менеджеру.

Для адміністративної частини визначено вимоги створення й редагування довідників, перегляду списку бронювань, зміни статусу, пошуку за клієнтом, туром або менеджером, фільтрації за статусом і датою, додавання платежів і туристичних документів у межах бронювання. Також потрібні групові дії для схвалення або скасування вибраних бронювань.

До нефункціональних вимог належать цілісність даних, простота супроводу, зрозуміла структура моделей, захист від несанкціонованого доступу, україномовний інтерфейс, масштабованість структури та можливість розширення. Важливо, щоб користувач бачив лише власні бронювання, а службові операції виконувалися через захищений адміністративний модуль.

Вимоги до якості програмного забезпечення можна співвіднести з моделлю ISO/IEC 25010, у якій виділяються функціональна придатність, надійність, зручність використання, продуктивність, супроводжуваність і безпека [6]. Для розробленої системи найбільш критичними є функціональна придатність, коректність даних, зручність адміністрування та можливість подальшого розвитку.

Оскільки система працює з персональними даними клієнтів, необхідно враховувати принцип мінімізації доступу. У базовій реалізації клієнт не має доступу до чужих бронювань, а зміна службових статусів винесена в адміністративну панель. Для подальшого промислового використання доцільно додати детальні ролі, журнал аудиту, політику складних паролів, резервне копіювання та захист завантажених документів [7, 20-21].

Сформульовані вимоги стали основою для проєктування бази даних. Вони визначили склад сутностей, типи зв'язків, правила видалення, набір статусів і структуру інтерфейсів. Подальші розділи показують, як ці вимоги були переведені у концептуальну, логічну й фізичну модель, а потім реалізовані засобами Django та PostgreSQL.

Вимоги до інтерфейсу сформульовано з урахуванням двох груп користувачів. Клієнту потрібні прості сторінки з мінімальною кількістю службових деталей. Працівнику, навпаки, потрібна щільна таблиця з фільтрами, пошуком і швидкою зміною статусів. Розділення цих інтерфейсів зменшує складність кожного сценарію.

Вимога до відтворюваності реалізації є важливою для дипломної роботи. Проєкт містить файл залежностей, міграції та команду наповнення демоданими. Завдяки цьому перевіряючий може повторити запуск системи й отримати ті самі екрани, що наведені у звіті.

Вимога до майбутнього розширення вплинула на те, що оплати й документи не зберігаються безпосередньо в бронюванні. Якщо в майбутньому

агентство використовуватиме часткові оплати або кілька документів на одну подорож, поточна структура уже підтримує такий сценарій.

Таблиця 1.4 – Функціональні та нефункціональні вимоги

Група	Вимога	Реалізація у проєкті
Функціональна	Перегляд активних турів	Сторінка каталогу, фільтр `active=True`
Функціональна	Пошук і фільтрація турів	Q-запити за назвою, країною, містом, готелем; фільтр за напрямом
Функціональна	Реєстрація клієнта	ClientRegistrationForm створює User і Client
Функціональна	Створення бронювання	BookingForm, автоматичне призначення менеджера, розрахунок суми
Функціональна	Заявка гостя	TourRequestForm і модель TourRequest
Функціональна	Адміністрування бронювань	Django Admin, list_editable, actions, filters
Нефункціональна	Цілісність даних	ForeignKey, OneToOneField, PROTECT, CASCADE, choices
Нефункціональна	Безпека доступу	login_required, CSRF, перевірка власника бронювання
Нефункціональна	Супроводжуваність	Модульна структура Django, міграції, демодані
Нефункціональна	Локалізація	Україномовні шаблони, verbose_name, LANGUAGE_CODE='uk'

Висновки до розділу 1

У першому розділі проаналізовано предметну область туристичного агентства, основні бізнес-процеси та наявні підходи до автоматизації внутрішнього обліку. Визначено склад даних, ролі користувачів, функціональні й нефункціональні вимоги, які стали основою для подальшого проєктування бази даних.

РОЗДІЛ 2
ПРОЄКТУВАННЯ БАЗИ ДАНИХ

2.1. Визначення функціональних та нефункціональних вимог

Проєктування бази даних починається з уточнення функціональних і нефункціональних вимог, оскільки саме вони визначають майбутню структуру таблиць. Якщо вимога сформульована нечітко, у моделі з'являються зайві поля або, навпаки, відсутні важливі зв'язки. Для туристичного агентства базовою вимогою є підтримка наскрізного обліку: від клієнта й туру до бронювання, платежу та документа.

Функціональні вимоги поділено на користувацькі та адміністративні. Користувацькі вимоги описують роботу клієнта з сайтом: перегляд турів, пошук, фільтрацію, реєстрацію, вхід, створення бронювання та перегляд власних бронювань. Адміністративні вимоги описують роботу працівника: створення турів, ведення довідників, редагування статусів, додавання платежів, документів і обробку заявок.

Нефункціональні вимоги впливають на технічні рішення. Вимога цілісності даних обумовила використання зовнішніх ключів і захисних правил видалення. Вимога зручності адміністрування призвела до використання Django Admin. Вимога супроводжуваності сприяла винесенню спільних полів `created_at` і `updated_at` в абстрактну модель TimestampedModel.

Важливою вимогою є локалізація. Користувацькі сторінки, назви моделей, поля форм і повідомлення реалізовані українською мовою. Це підвищує зрозумілість системи для цільових користувачів і відповідає вимозі методичних рекомендацій щодо виконання роботи державною мовою [1].

Вимога безпеки реалізована базовими механізмами Django: автентифікацією, декораторами `login_required`, CSRF-захистом у формах і перевіркою належності бронювання конкретному клієнту. У методі скасування бронювання запис шукається не лише за ідентифікатором, а й за клієнтом

поточного користувача. Це запобігає зміні чужого бронювання через підстановку іншого номера в URL.

Вимога розширюваності відображена в тому, що сутності не злиті в одну велику таблицю. Наприклад, платежі й документи винесені окремо від бронювання. Завдяки цьому в майбутньому можна додати кілька платежів, різні типи документів, файли, статуси перевірки або експорт без зміни базової структури бронювання.

Уточнення вимог також допомогло відокремити обов'язкові функції від перспективних. До обов'язкових належать довідники, бронювання, платежі, документи, заявки та базова автентифікація. До перспективних віднесено аналітичні звіти, історію статусів, резервування залишку місць і деталізовані ролі працівників.

Нефункціональні вимоги часто не видно на скріншотах, але саме вони визначають якість системи. Наприклад, правило `PROTECT` не створює окремої сторінки, однак запобігає випадковому руйнуванню історії бронювань. Аналогічно, `select_related` не змінює інтерфейс, але впливає на ефективність роботи сторінки.

Вимоги до якості даних мають особливе значення, оскільки туристичне агентство працює з контактами й документами клієнтів. Неправильна електронна адреса, некоректний номер паспорта або помилковий статус можуть спричинити службові проблеми. Тому частина контролю винесена у форми та обмеження моделей.

2.2. Побудова концептуальної моделі даних

Концептуальна модель описує предметну область на рівні сутностей та їхніх зв'язків без прив'язки до конкретної СУБД. У межах цієї роботи виділено дев'ять сутностей: Client, Employee, Destination, Hotel, Tour, Booking, Payment, TravelDocument і TourRequest. Вони охоплюють довідкові, операційні, фінансові й комунікаційні аспекти діяльності туристичного агентства.

Центральною сутністю є Booking. Вона поєднує клієнта, тур і менеджера, а також виступає основою для платежів і документів. Така структура відповідає реальному процесу: спочатку виникає домовленість щодо конкретного туру, потім вона оформлюється як бронювання, після чого можуть додаватися платежі, договір, ваучер, страховка або квитки.

Сутність Client відображає туриста або замовника послуги. У системі вона пов'язана з обліковим записом Django User через зв'язок один-до-одного. Це дає змогу використовувати стандартну автентифікацію фреймворку, але зберігати предметно-орієнтовані дані клієнта окремо від технічного запису користувача.

Сутність Employee відображає працівника агентства. Вона має поле position, яке обмежене набором значень: менеджер, бухгалтер, директор. Наявність поля active дозволяє виключати неактивних працівників із автоматичного призначення на нові бронювання без видалення їх з історії системи.

Destination і Hotel утворюють довідкову частину туристичного продукту. Напрям містить країну й місто або курорт. Готель належить до напряму і може використовуватися в одному або кількох турах. Така декомпозиція зменшує дублювання і дає змогу фільтрувати тури за напрямом.

Tour описує конкретну пропозицію з датами, місцями, ціною і типом харчування. У концептуальній моделі тур пов'язується з напрямом обов'язково, а з готелем опціонально. Це важливо, оскільки не кожний туристичний продукт передбачає проживання в конкретному готелі, наприклад екскурсійний тур або індивідуальна заявка.

Payment і TravelDocument є залежними сутностями щодо Booking. Платежів може бути декілька, адже клієнт може вносити передоплату й остаточний платіж окремо. Документів також може бути декілька, оскільки комплект туриста зазвичай складається з договору, ваучера, страхового поліса та квитків.

TourRequest описує попередню заявку з сайту. Вона може бути пов'язана з конкретним туром, але не обов'язково. Це дозволяє використовувати одну форму як для заявки на конкретну пропозицію, так і для індивідуального підбору подорожі. Таким чином, система підтримує клієнта на етапі, коли бронювання ще не створене.

Концептуальна модель також відображає різницю між довідковими й транзакційними сутностями. Destination, Hotel і Employee здебільшого описують стан системи, тоді як Booking, Payment і TourRequest описують події. Такий поділ полегшує розуміння моделі та пояснює, чому центральним елементом обліку є бронювання.

Зв'язок між Tour і Hotel зроблено необов'язковим, оскільки не всі туристичні продукти мають готельну складову. Це рішення підвищує гнучкість моделі: у майбутньому можна додати екскурсійні або консультаційні послуги без примусового створення фіктивного готелю.

TourRequest не замінює Client, оскільки заявка може надійти від людини, яка ще не зареєстрована. Якщо примусово створювати клієнта для кожної заявки, у довіднику клієнтів швидко з'являться неперевірені записи. Окрема таблиця дозволяє спочатку опрацювати звернення, а потім, за потреби, створити клієнтський профіль.

На рисунку 2.1 подано узагальнену ER-модель, яка показує центральне місце бронювання та зв'язки між довідковими, операційними, фінансовими й документальними сутностями.

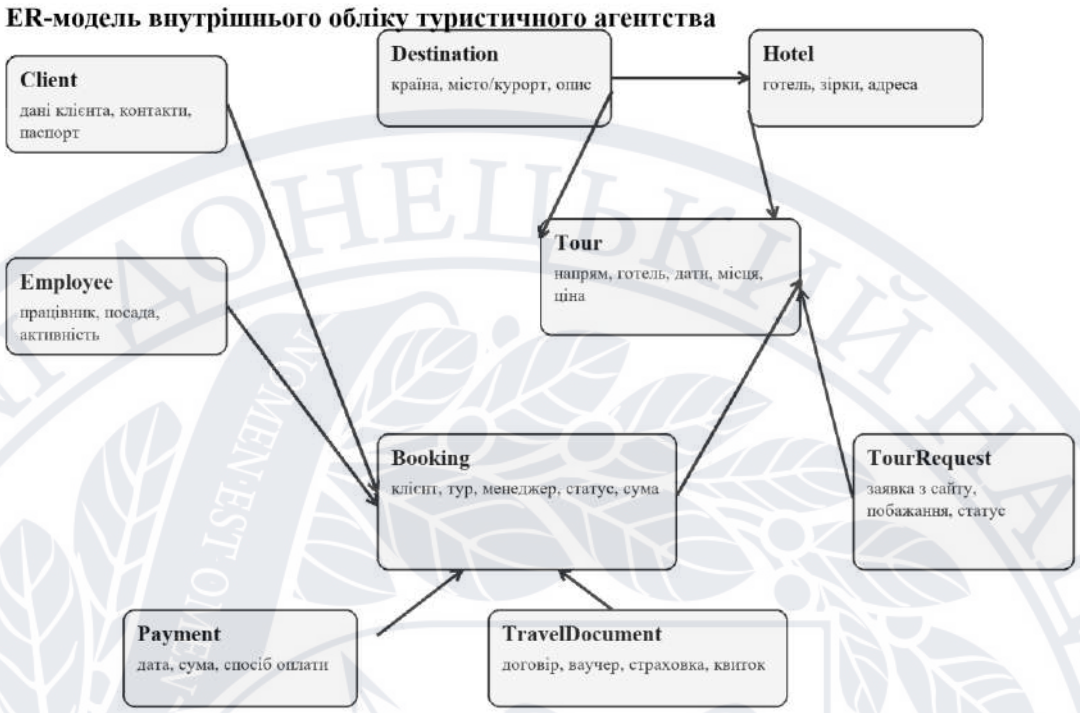

Рисунок 2.1 – Концептуальна ER-модель бази даних туристичного агентства

Таблиця 2.1 – Сутності концептуальної моделі

Сутність	Призначення	Основні зв'язки
Client	Зберігає дані клієнта та зв'язок з обліковим записом	1:N з Booking, 1:1 з User
Employee	Описує працівників агентства	1:N з Booking
Destination	Довідник країн і міст	1:N з Hotel, 1:N з Tour
Hotel	Довідник готелів у межах напряму	N:1 з Destination, 1:N з Tour
Tour	Туристична пропозиція з датами і ціною	N:1 з Destination, N:1 з Hotel, 1:N з Booking
Booking	Центральна операція обліку	N:1 з Client, Tour, Employee; 1:N з Payment і TravelDocument
Payment	Фінансовий запис за бронюванням	N:1 з Booking
TravelDocument	Документальний супровід бронювання	N:1 з Booking
TourRequest	Попередня заявка з сайту	N:1 з Tour або без прив'язки

2.3. Логічне проєктування бази даних

Логічне проєктування полягає у перетворенні концептуальної моделі на набір відношень із ключами, атрибутами й обмеженнями. У роботі використано реляційний підхід, за якого кожна сутність предметної області відповідає окремій таблиці, а зв'язки між сутностями реалізуються через зовнішні ключі. Такий підхід є класичним для інформаційних систем облікового типу [22-24], [25].

Першою нормальною формою забезпечено атомарність значень. Наприклад, дані клієнта розділено на `first_name`, `last_name`, `phone`, `email`, `passport_number`, а не збережено одним текстовим полем. Дати туру зберігаються у двох окремих полях `start_date` і `end_date`, що дозволяє виконувати фільтрацію й сортування.

Друга нормальна форма досягається завдяки тому, що неключові атрибути залежать від первинного ключа своєї таблиці. Наприклад, назва готелю, кількість зірок і адреса залежать від готелю, а не від туру або бронювання. У таблиці Booking зберігаються лише атрибути, що характеризують конкретне бронювання.

Третя нормальна форма підтримується винесенням довідкових даних у самостійні таблиці. Країна й місто не дублюються в кожному турі, а належать сутності Destination. Готель не дублюється в бронюванні, а отримується через зв'язок з туром. Це зменшує ризик неузгоджених записів, наприклад різного написання одного й того самого міста.

Для контролю допустимих значень у моделі використано поля з choices. Статус бронювання може бути новим, підтвердженим, оплаченим або скасованим. Спосіб оплати може бути готівкою, карткою або банківським переказом. Тип документа обмежено договором, ваучером, страховкою або квитком. Обмежені набори значень спрощують інтерфейс і зменшують помилки введення.

Унікальні обмеження застосовано для напрямів і готелів. Комбінація країни та міста в Destination має бути унікальною, а комбінація напряму й назви

готелю - також. Це важливо для чистоти довідників: якщо однаковий готель буде створено кілька разів, тури й звіти стануть менш надійними.

Під час логічного проєктування також враховано правила видалення. Для зв'язків бронювання з клієнтом, туром і працівником використовується захист від видалення, оскільки видалення цих об'єктів може порушити історію операцій. Для платежів і документів, навпаки, використано каскадне видалення від бронювання, оскільки вони не мають самостійного сенсу без нього.

Логічна модель зберігає баланс між нормалізацією й практичністю. Вона не містить зайвих проміжних таблиць там, де зв'язок є простим один-до-багатьох, але водночас виділяє всі сутності, що мають власну життєву роль у предметній області. Це робить модель зрозумілою для супроводу й достатньо гнучкою для майбутнього розвитку.

Нормалізація також полегшує оновлення даних. Якщо в одному готелі зміниться адреса або категорія, достатньо оновити один запис Hotel, а не всі тури, де він згадується. Це знижує ймовірність суперечливих даних і робить довідники надійнішими.

Використання choice-полів є компромісом між повною нормалізацією і практичністю. Статуси можна було б винести в окремі таблиці, але для невеликого набору сталих значень зручніше використовувати перелік у моделі. Це спрощує код і не шкодить цілісності даних.

Логічна модель залишає можливість майбутнього додавання історії статусів. Нині статус зберігається в таблиці Booking як поточний стан. Якщо буде потрібно відстежувати всі зміни, можна додати таблицю BookingStatusHistory із посиланням на бронювання, датою, попереднім і новим статусом.

Таблиця 2.2 – Рішення нормалізації моделі даних

Рішення	Пояснення	Очікуваний ефект
Винесення Destination	Країна й місто зберігаються окремо від туру	Зменшення дублювання напрямів

Рішення	Пояснення	Очікуваний ефект
Винесення Hotel	Готель належить напряму й може використовуватися в кількох турах	Єдиний довідник готелів
Окрема таблиця Payment	Одне бронювання може мати кілька платежів	Підтримка часткової оплати
Окрема таблиця TravelDocument	До бронювання може належати кілька документів	Документальний облік без перевантаження Booking
Окрема таблиця TourRequest	Заявка не є бронюванням до підтвердження	Чистий облік потенційних клієнтів
Choices для статусів	Статуси обмежені переліком допустимих значень	Менше помилок введення
PROTECT для історичних зв'язків	Клієнт, тур і працівник не видаляються, якщо є бронювання	Збереження історії операцій

2.4. Фізичне проєктування бази даних

Фізичне проєктування визначає, як логічна модель реалізується у конкретній технологічній платформі. У проєкті фізичну модель створено засобами Django ORM, а зберігання даних виконується у PostgreSQL. Такий підхід дозволяє описувати таблиці у вигляді Python-класів, а структуру бази даних створювати й змінювати через міграції [9-11].

Кожна модель Django відповідає таблиці в базі даних. Типи полів обрано відповідно до характеру даних: `CharField` для коротких текстових значень, `TextField` для коментарів і описів, `EmailField` для електронної пошти, `DateField` для дат, `DecimalField` для сум, `BooleanField` для логічних ознак, `ForeignKey` і `OneToOneField` для зв'язків між таблицями.

Грошові значення реалізовано через `DecimalField`, а не через числа з плаваючою комою. Це важливо для фінансового обліку, оскільки операції з оплатами потребують точності. У таблицях Tour і Booking поля цін мають `max_digits=10` та `decimal_places=2`, що дозволяє зберігати суми у гривнях з копійками.

Мітки часу `created_at` і `updated_at` винесено в абстрактну модель TimestampedModel. Від неї успадковуються предметні сутності, тому кожний

запис має інформацію про момент створення й оновлення. Це корисно для аудиту, пошуку нових заявок і майбутньої аналітики змін.

Фізична реалізація враховує можливість роботи з файлами туристичних документів. Модель TravelDocument містить поле `FileField`, яке зберігає шлях до завантаженого файла. У демонстраційному проєкті це поле є опціональним, але його наявність показує, як система може бути розширена до реального документообігу.

У налаштуваннях проєкту база даних задається через змінні середовища. Це дозволяє відокремити код від конфігурації й змінювати параметри підключення без редагування вихідних файлів. Для навчального запуску передбачено значення за замовчуванням, а для практичного середовища можна вказати власні `POSTGRES_DB`, `POSTGRES_USER`, `POSTGRES_PASSWORD`, `POSTGRES_HOST` і `POSTGRES_PORT`.

У фізичній моделі важливо враховувати індексацію. Django і PostgreSQL автоматично створюють індекси для первинних ключів та зовнішніх ключів. Для подальшого збільшення обсягу даних доцільно додати індекси на поля, що часто використовуються у фільтрації: `status`, `booking_date`, `start_date`, `payment_date`, а також на комбінації, пов'язані з пошуком бронювань.

Фізичне проєктування завершилося створенням міграцій, які є версійним описом змін структури бази даних. Міграції дозволяють повторити створення таблиць на іншому комп'ютері, синхронізувати структуру між середовищами й документувати еволюцію моделі. Для дипломного проєкту це важливо, оскільки структура бази даних не є прихованою частиною програми, а може бути відтворена й перевірена.

Django ORM не приховує реляційну природу бази даних, а надає зручний шар опису. Кожен ForeignKey у моделі створює реальний зовнішній ключ у PostgreSQL. Це означає, що цілісність підтримується не лише кодом застосунку, а й самою базою даних.

Міграції виконують роль технічної документації змін. Якщо структура моделі змінюється, створюється новий файл міграції, який можна переглянути, застосувати або відкотити. Для дипломного проєкту це зручно, адже процес розробки стає відтворюваним і контрольованим.

Фізична реалізація також враховує локалізацію. У налаштуваннях вказано `LANGUAGE_CODE = 'uk'` і часовий пояс `Europe/Kyiv`. Це робить відображення дат і системних повідомлень ближчим до очікувань українського користувача.

На рисунку 2.2 показано фізичну архітектуру рішення: клієнтський браузер і адміністративна панель працюють із серверною логікою Django, а збереження даних виконується у PostgreSQL через ORM.

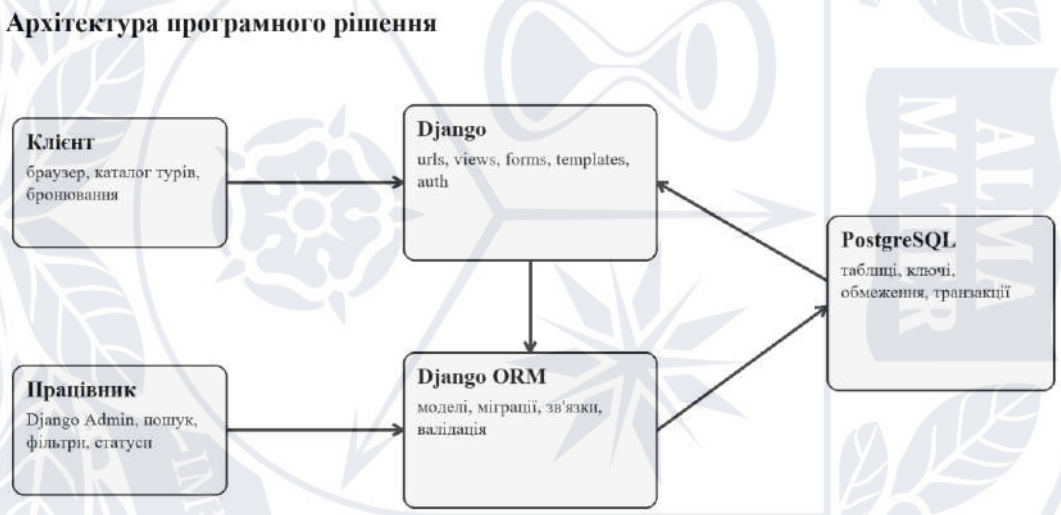

Рисунок 2.2 – Архітектура реалізованої інформаційної системи

Таблиця 2.3 – Відповідність типів даних Django і призначення полів

Тип поля	Приклад у проєкті	Призначення
CharField	title, first_name, phone	Короткі текстові значення з обмеженням довжини
TextField	description, notes, comment	Розширені коментарі та описи
EmailField	email	Електронна адреса з базовою перевіркою формату

Тип поля	Приклад у проєкті	Призначення
DateField	start_date, payment_date, issued_at	Дати турів, платежів і документів
DecimalField	base_price, total_amount, amount	Фінансові значення з фіксованою точністю
BooleanField	active	Логічна ознака активності запису
ForeignKey	tour, client, employee, booking	Зв'язки один-до-багатьох між таблицями
OneToOneField	Client.user	Зв'язок клієнтського профілю з користувачем Django
FileField	TravelDocument.file	Прикріплення файла туристичного документа

2.5. Розробка структури таблиць, індексів, обмежень

Структура таблиць розроблена так, щоб кожна таблиця мала чітке призначення. Довідкові таблиці Destination, Hotel, Employee і частково Client забезпечують стабільну основу. Операційна таблиця Booking фіксує основні дії клієнтів. Таблиці Payment і TravelDocument деталізують фінансовий і документальний супровід. TourRequest зберігає попередні звернення, які ще не стали бронюваннями.

Первинні ключі створюються автоматично як BigAutoField, що відповідає типовій конфігурації Django. Зовнішні ключі забезпечують зв'язки між таблицями і дозволяють ORM ефективно отримувати пов'язані об'єкти через `select_related` та `prefetch_related`. У представленнях це використовується для зменшення кількості SQL-запитів під час виведення списків турів і бронювань.

Обмеження цілісності реалізовані на кількох рівнях. На рівні моделі задано обов'язковість полів, максимальні довжини, набори допустимих значень і унікальність окремих комбінацій. На рівні форм додано перевірку кількості туристів, яка не може бути меншою за одиницю. На рівні представлень перевіряється автентифікація користувача та належність бронювання поточному клієнту.

У таблиці Booking поле `total_amount` обчислюється під час створення бронювання як добуток базової ціни туру та кількості туристів. Це спрощує подальшу фінансову звітність, оскільки сума бронювання зберігається в операційному записі. Водночас у майбутньому можна додати механізм перерахунку або історію зміни ціни, якщо система буде використовуватися в реальному бізнесі.

У таблиці Tour поле `active` дозволяє не видаляти завершені або тимчасово недоступні тури, а просто приховувати їх із публічного каталогу. Це важливе рішення для збереження історії: бронювання на старий тур можуть залишатися в базі, але клієнти не бачитимуть його серед актуальних пропозицій.

Таблиця TourRequest має окремий статус опрацювання. Це дозволяє відокремити потенційний продаж від підтвердженого бронювання. Менеджер може спочатку зв'язатися з клієнтом, уточнити умови, а вже потім створити бронювання. Такий підхід відповідає реальному процесу туристичного агентства, де не кожна заявка одразу завершується продажем.

Узагальнена структура таблиць показує, що модель є достатньо компактною, але покриває основні сценарії внутрішнього обліку. Вона може бути розширена додаванням таблиць для договорів із туроператорами, знижок, промокодів, валют, історії статусів, комісій менеджерів і звітів. При цьому наявна структура не потребує радикальної перебудови.

Словник даних є важливою частиною пояснення моделі, оскільки дозволяє побачити не лише назви таблиць, а й призначення конкретних полів. У межах звіту він поданий у стислому вигляді для ключових атрибутів, що визначають роботу системи.

Під час розвитку системи доцільно створити окремі індекси для пошукових полів. Наприклад, пошук клієнта за прізвищем або телефоном може стати частою дією менеджера. У демонстраційній базі це не критично, але в реальній експлуатації індексація безпосередньо впливатиме на швидкість роботи.

Структура таблиць також підготовлена до звітності. Наявність дат створення, дат бронювання, дат платежів, статусів і відповідальних працівників дозволяє будувати агреговані показники без додавання нових полів. Це підтверджує, що модель придатна не лише для введення даних, а й для подальшого аналізу.

Таблиця 2.4 – Словник даних основних таблиць

Таблиця	Поле	Тип	Призначення	Обмеження
Client	user	OneToOneField	зв'язок із користувачем	CASCADE, optional
Client	first_name, last_name	CharField	ім'я та прізвище клієнта	required
Client	phone, email	CharField / EmailField	контакти клієнта	phone required
Client	passport_number	CharField	паспортні дані	optional
Employee	position	CharField choices	посада працівника	manager/accountant/director
Employee	active	BooleanField	ознака чинного працівника	default True
Destination	country, city	CharField	географічний напрям	unique together
Hotel	destination	ForeignKey	напрям готелю	PROTECT
Hotel	name, stars	CharField / PositiveSmallIntegerField	назва і категорія	unique with destination
Tour	destination	ForeignKey	напрям туру	PROTECT
Tour	hotel	ForeignKey	готель туру	PROTECT, optional
Tour	start_date, end_date	DateField	період подорожі	required
Tour	seats_total	PositiveIntegerField	загальна кількість місць	required
Tour	base_price	DecimalField	базова ціна туру	10,2
Tour	meal_type	CharField choices	тип харчування	limited values
Tour	active	BooleanField	публікація у каталозі	default True
Booking	client	ForeignKey	клієнт бронювання	PROTECT

Таблиця	Поле	Тип	Призначення	Обмеження
Booking	tour	ForeignKey	обраний тур	PROTECT
Booking	employee	ForeignKey	відповідальний менеджер	PROTECT
Booking	tourists_count	PositiveIntegerField	кількість туристів	min 1 via form
Booking	total_amount	DecimalField	сума бронювання	computed on create
Booking	status	CharField choices	стан бронювання	new/confirmed/ paid/cancelled
Payment	booking	ForeignKey	належність до бронювання	CASCADE
Payment	payment_date	DateField	дата оплати	required
Payment	amount	DecimalField	сума платежу	10,2
Payment	method	CharField choices	спосіб оплати	cash/card/transf er
TravelDocum ent	booking	ForeignKey	документ бронювання	CASCADE
TravelDocum ent	document_type	CharField choices	тип документа	contract/vouche r/insurance/tick et
TravelDocum ent	number	CharField	номер документа	required
TravelDocum ent	issued_at	DateField	дата видачі	required
TourRequest	tour	ForeignKey	пов'язаний тур	SET_NULL, optional
TourRequest	first_name, last_name	CharField	контактна особа	required
TourRequest	desired_country	CharField	бажана країна	optional
TourRequest	tourists_count	PositiveIntegerField	кількість туристів	default 1
TourRequest	status	CharField choices	стан опрацювання заявки	new/contacted/b ooked/closed

Висновки до розділу 2

У другому розділі на основі сформульованих вимог побудовано концептуальну, логічну та фізичну модель бази даних туристичного агентства. Визначено сутності, зв'язки, ключі, обмеження та структуру основних таблиць, що забезпечують цілісне зберігання даних про клієнтів, тури, бронювання, оплати, документи й заявки.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ БАЗИ ДАНИХ

3.1. Створення бази даних та програмної частини

Програмну частину реалізовано як Django-проєкт `config` із застосунком `agency`. У застосунку зосереджені моделі предметної області, форми, маршрути, представлення, шаблони, статичні стилі та команда наповнення демонстраційними даними. Така структура відповідає типовій організації Django-проєктів і полегшує супровід коду.

Файл `models.py` містить опис усіх сутностей бази даних. Саме він є основою для міграцій, адміністративної панелі та форм. Моделі мають україномовні `verbose_name` і `verbose_name_plural`, завдяки чому адміністративний інтерфейс відображає назви сутностей зрозуміло для користувача.

Файл `admin.py` налаштовує роботу Django Admin. Для кожної моделі визначено поля списку, пошук, фільтри, автодоповнення і, за потреби, ієрархію дат. Для бронювань додано inline-редагування платежів і документів, а також дії для схвалення та скасування вибраних записів. Це перетворює стандартну адмін-панель на повноцінний службовий інструмент.

Файл `forms.py` містить форми входу, реєстрації, створення бронювання та заявки. Реєстраційна форма створює не лише стандартного користувача Django, а й пов'язаний запис Client. Це важливо для узгодження технічної автентифікації з предметною моделлю обліку клієнтів.

Файл `views.py` реалізує серверну логіку сторінок. Представлення `tour_list` формує каталог активних турів із пошуком і фільтрацією. `tour_detail` показує деталі туру і залежно від стану авторизації відображає форму бронювання або заявку гостя. `my_bookings` виводить бронювання поточного клієнта, а `booking_cancel` дозволяє скасувати власний запис.

Маршрути в `urls.py` відображають основні сценарії: головна сторінка, реєстрація, вхід, вихід, особистий кабінет, скасування бронювання, створення

бронювання, детальна сторінка туру і загальна форма заявки. Такий набір URL є достатнім для демонстрації повного циклу роботи системи.

Шаблони HTML описують користувацький інтерфейс. Базовий шаблон містить навігацію, блок повідомлень і підключення стилів. Сторінки каталогу, деталей туру, входу, реєстрації, заявки та бронювань успадковують базовий шаблон. Це зменшує дублювання і забезпечує однакову структуру сторінок.

Стилі в `styles.css` формують візуальну частину клієнтського інтерфейсу: сітку карток турів, героїчний блок, панель пошуку, форми, картки бронювань і адаптивне розміщення елементів. Для дипломного проєкту це демонструє, що база даних не є абстрактною, а має реальний інтерфейс взаємодії з користувачем.

Команда `seed_demo` створює тестові акаунти, працівників, клієнтів, напрями, готелі, тури, бронювання й платежі. Це важливо для відтворюваності демонстрації: після розгортання проєкту можна швидко отримати наповнену базу й перевірити роботу інтерфейсів без ручного введення десятків записів.

Архітектура Django-проєкту є зрозумілою для супроводу. Функції маршрутизації, представлення, форм, моделей і шаблонів розділено по файлах. Завдяки цьому зміна форми бронювання не потребує редагування моделей, а оновлення шаблону не впливає на структуру бази даних.

Адміністративна панель стала ключовим інструментом внутрішнього обліку. У ній працівник бачить не технічні назви таблиць, а україномовні сутності. Пошук і фільтри зменшують час на знаходження потрібного запису, а inline-блоки платежів і документів зберігають контекст бронювання.

Користувацький інтерфейс не дублює функції адміністратора. Клієнт бачить лише те, що потрібно для вибору туру й роботи зі своїми бронюваннями. Такий підхід спрощує навігацію й зменшує ризик випадкового доступу до службових даних.

Таблиця 3.1 – Основні файли програмної реалізації

Файл	Призначення	Ключові елементи
agency/models.py	Опис структури бази даних	Client, Employee, Destination, Hotel, Tour, Booking, Payment, TravelDocument, TourRequest
agency/admin.py	Налаштування службового інтерфейсу	list_display, filters, search, inlines, actions
agency/forms.py	Форми користувача	ClientRegistrationForm, BookingForm, TourRequestForm
agency/views.py	Серверна логіка сторінок	tour_list, tour_detail, my_bookings, booking_cancel
agency/urls.py	Маршрути застосунку	каталог, вхід, реєстрація, бронювання, заявка
templates/agency	HTML-шаблони клієнтського сайту	base, tour_list, tour_detail, my_bookings, request_form
static/agency/styles.css	Оформлення інтерфейсу	картки турів, форми, кабінет, адаптивність
management/commands/seed_demo.py	Наповнення демоданими	користувачі, працівники, тури, бронювання, платежі
config/settings.py	Конфігурація проєкту	PostgreSQL, мова, часова зона, auth redirects

На рисунку 3.1 наведено головну сторінку користувацького інтерфейсу. Вона демонструє каталог активних турів, пошукову форму, фільтр напрямів і картки пропозицій із ключовими параметрами.

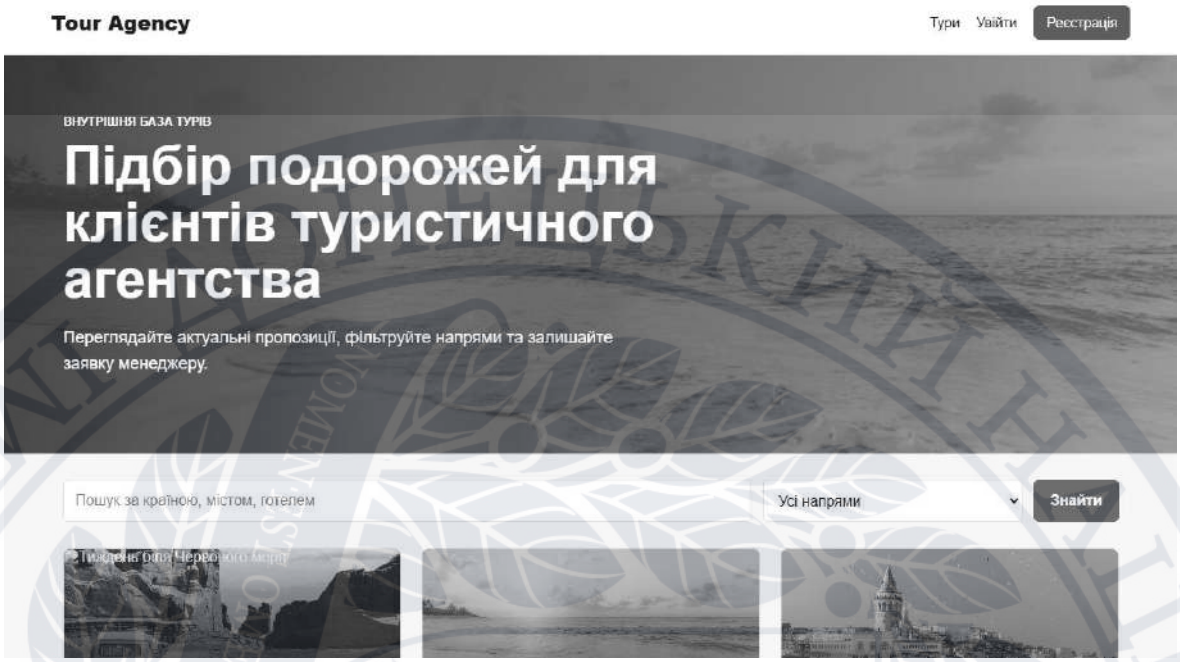

Рисунок 3.1 – Головна сторінка користувацького веб-інтерфейсу

На рисунку 3.2 показано сторінку детального перегляду туру для гостя. Інтерфейс містить службову інформацію про тур і форму заявки, оскільки неавторизований користувач не може створити пряме бронювання.

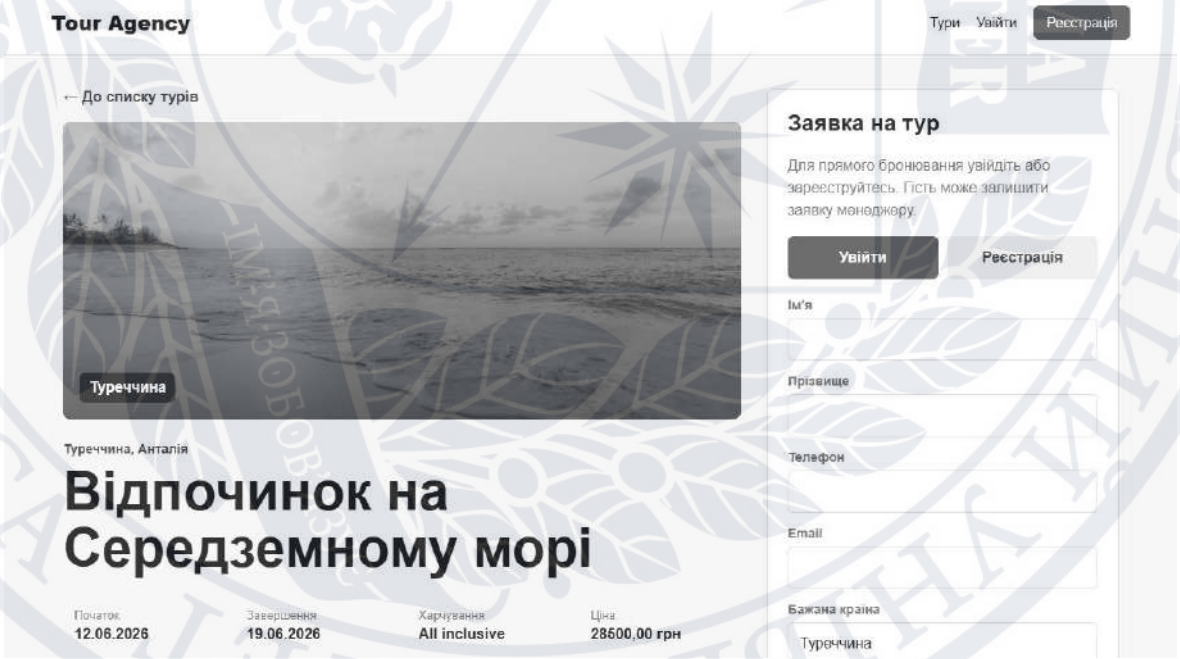

Рисунок 3.2 – Детальна сторінка туру для неавторизованого користувача

3.2. Реалізація операцій внутрішнього обліку

Операції внутрішнього обліку реалізовано через поєднання користувацького сайту і Django Admin. Клієнтська частина відповідає за

перегляд і створення первинних даних, а адміністративна - за службову обробку. Такий розподіл відповідає реальному процесу туристичного агентства, де клієнт ініціює запит, а працівник підтверджує умови.

Під час перегляду каталогу система отримує лише активні тури. Для оптимізації використовується `select_related` для завантаження напряму й готелю разом із туром. Це зменшує кількість окремих SQL-запитів і робить сторінку придатною для виведення багатьох карток.

Пошук у каталозі реалізовано через об'єкти `Q`, що дозволяють шукати за назвою туру, країною, містом і назвою готелю. Окремо доступна фільтрація за напрямом. Такий підхід є простим, але достатнім для навчального прототипу, оскільки покриває найтиповіші дії клієнта під час вибору подорожі.

Створення бронювання доступне лише авторизованому користувачу. Якщо користувач успішно надсилає форму, система знаходить або створює пов'язаний запис Client, призначає активного менеджера, встановлює тур, обчислює суму й зберігає бронювання зі статусом «Нове». Після цього користувач перенаправляється до сторінки власних бронювань.

Якщо відвідувач не авторизований, детальна сторінка туру показує форму заявки. Така заявка не створює бронювання, але фіксує контактні дані й побажання клієнта. Це дозволяє агентству працювати з потенційними клієнтами, які ще не готові проходити реєстрацію або не впевнені у виборі туру.

Особистий кабінет відображає бронювання поточного клієнта. У запиті використовуються `select_related` і `prefetch_related`, щоб завантажити тур, напрям, готель, працівника, платежі й документи. Скасовані бронювання виключаються зі списку, тому сторінка показує лише актуальні записи.

Скасування бронювання реалізовано через POST-запит із CSRF-захистом. Система перевіряє, що бронювання належить поточному клієнту. Якщо воно ще не скасоване, статус змінюється на `cancelled`, а дата оновлення фіксується автоматично. Такий сценарій не видаляє запис із бази, а зберігає історію операції.

В адміністративній панелі менеджер бачить список бронювань із клієнтом, туром, менеджером, датою, кількістю туристів, сумою і статусом. Поле статусу редагується прямо у списку. Крім того, доступні групові дії для схвалення та скасування. Це зручно для службової обробки багатьох заявок.

Платежі й документи редагуються через inline-форми в картці бронювання. Працівник може бачити фінансовий та документальний супровід не в окремих списках, а в контексті конкретного бронювання. Це відповідає предметній логіці, де платіж і документ існують як частина обслуговування певного туру.

Автоматичне призначення менеджера реалізовано через вибір активного працівника з посадою менеджера. Якщо таких працівників немає, система може призначити будь-якого активного працівника або показати повідомлення про неможливість оформлення. Це демонструє обробку нештатної ситуації на рівні бізнес-логіки.

Повідомлення користувачу відіграють важливу роль у сценаріях. Після створення бронювання або заявки система не залишає користувача без зворотного зв'язку, а показує коротке пояснення подальших дій. Це підвищує зрозумілість інтерфейсу і зменшує кількість повторних натискань.

Логіка заявок і бронювань навмисно розділена. Гість залишає заявку, а авторизований клієнт створює бронювання. Це дозволяє системі коректно працювати з різним рівнем готовності користувача до купівлі й не змішувати попередні звернення з операційними записами.

Життєвий цикл бронювання у розробленій системі подано на рисунку 3.3. Він починається з вибору туру клієнтом і завершується відображенням актуального статусу в особистому кабінеті.

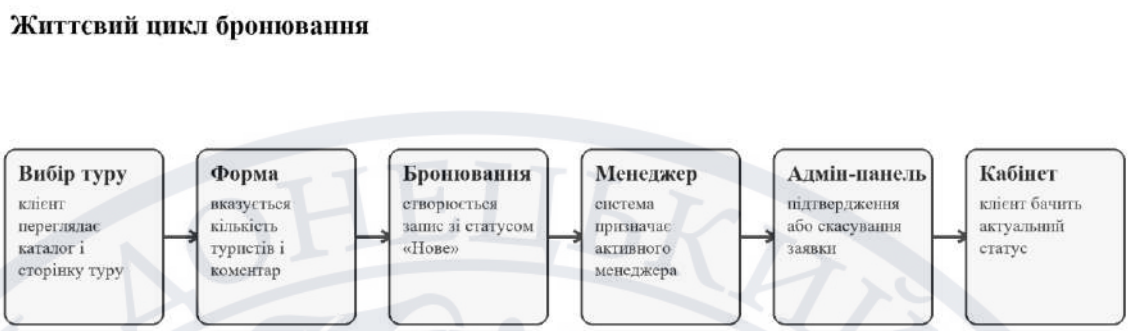

Рисунок 3.3 – Життєвий цикл бронювання у створеній системі

На рисунку 3.4 показано форму входу. Вона використовує стандартний механізм автентифікації Django, але відображає поля українською мовою й пояснює клієнту призначення особистого кабінету.

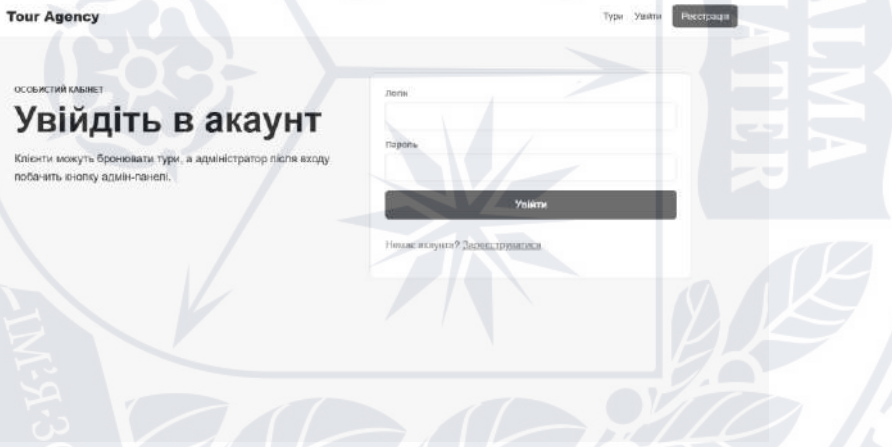

Рисунок 3.4 – Сторінка входу до системи

Після входу клієнт може створити бронювання безпосередньо на сторінці туру. Відповідний екран наведено на рисунку 3.5.

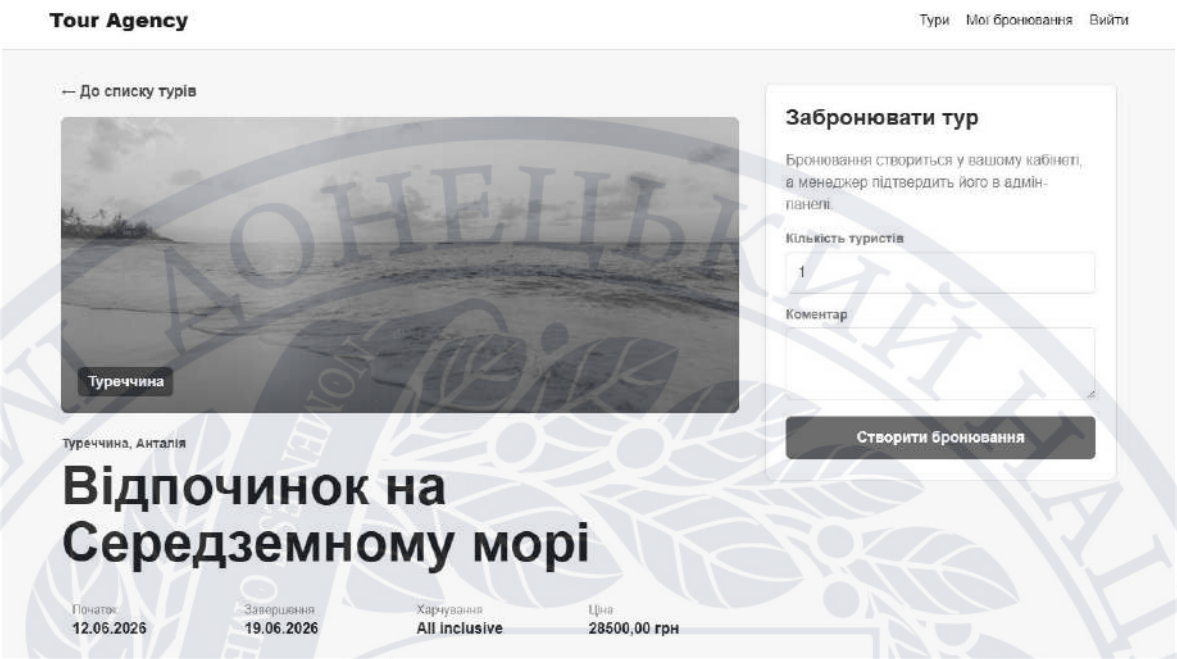

Рисунок 3.5 – Форма створення бронювання авторизованим клієнтом

Особистий кабінет клієнта, показаний на рисунку 3.6, відображає активні бронювання, їхній статус, суму, кількість туристів і відповідального менеджера.

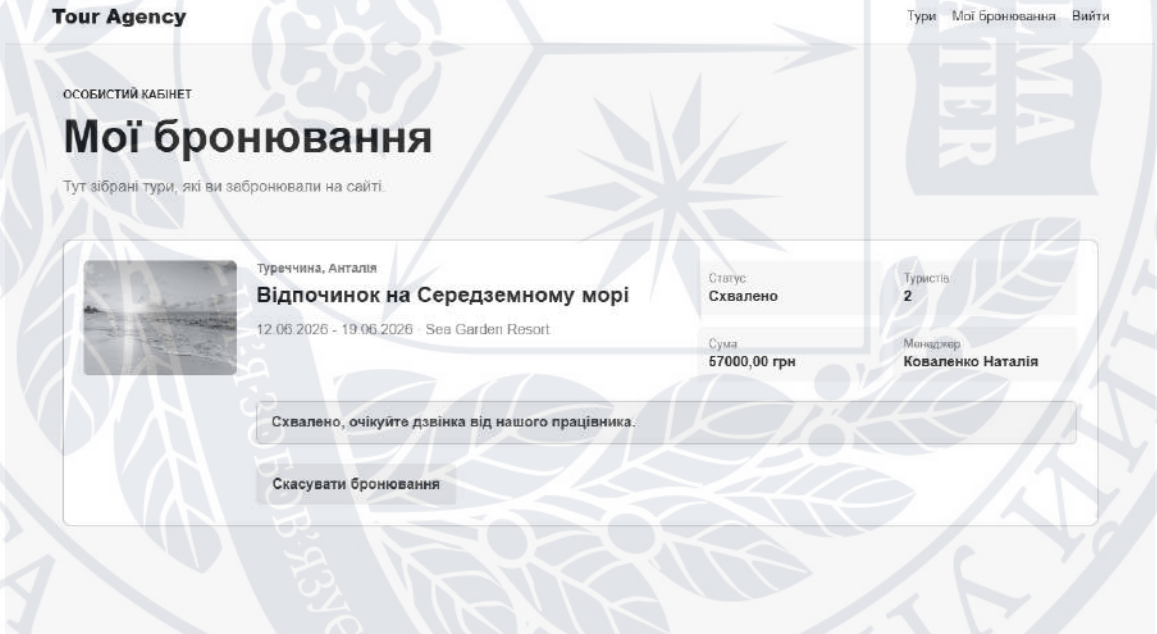

Рисунок 3.6 – Сторінка «Мої бронювання»

3.3. Тестування працездатності системи

Тестування працездатності системи виконано за основними користувацькими й адміністративними сценаріями. Метою тестування було перевірити, чи відповідає реалізація сформульованим вимогам, чи коректно

працюють зв'язки між сутностями та чи не виникають помилки при переході між ключовими сторінками.

Першим перевірено сценарій перегляду каталогу. Після запуску сервера головна сторінка відображає активні тури з країною, містом, готелем, датами, кількістю місць і ціною. Пошук за країною, містом або готелем звужує перелік результатів. Фільтр за напрямом дозволяє отримати тури конкретної країни або курорту.

Другим перевірено сценарій гостя. Неавторизований користувач може відкрити детальну сторінку туру, побачити інформацію про дати, харчування, ціну й готель, але замість прямого бронювання отримує форму заявки. Після надсилання форма створює запис TourRequest і показує повідомлення про успішне надсилання.

Третім перевірено сценарій реєстрації та входу. Після реєстрації створюється користувач Django і пов'язаний запис Client. Авторизований клієнт бачить у навігації пункт «Мої бронювання» і може створювати бронювання. Це підтверджує коректну взаємодію стандартної системи користувачів Django з предметною моделлю клієнта.

Четвертим перевірено створення бронювання. Клієнт обирає тур, вказує кількість туристів і коментар. Система створює Booking, автоматично обчислює суму, призначає менеджера і встановлює статус «Нове». Після створення запис відображається в особистому кабінеті з інформацією про менеджера й суму.

П'ятим перевірено скасування бронювання клієнтом. Після натискання кнопки скасування форма надсилає POST-запит, статус змінюється на «Скасовано», а запис прибирається з активного списку. При цьому запис залишається в базі даних і може бути переглянутий адміністратором.

Шостим перевірено адміністративний сценарій. Після входу в Django Admin працівник може переглядати моделі, фільтрувати бронювання, змінювати статус, додавати платежі й документи. Схвалення бронювання через дію адміністратора змінює відображення в особистому кабінеті клієнта.

Сьомим перевірено обробку помилкових даних. Форма бронювання не приймає кількість туристів менше одиниці. Система також не створює бронювання, якщо немає активного менеджера для призначення. У такому випадку користувач отримує повідомлення про неможливість оформлення, що запобігає створенню неповного запису.

Результати тестування показали, що базові сценарії працюють коректно, а модель даних забезпечує потрібні зв'язки. Виявлені напрями розвитку стосуються не помилок, а функціонального розширення: детальнішого контролю залишку місць, історії зміни статусів, ролей працівників і звітів за платежами.

Під час тестування також перевірено, що адміністративна панель відображає пов'язані поля у зручному вигляді. Наприклад, у списку бронювань видно клієнта, тур, менеджера, дату, кількість туристів, суму і статус. Це підтверджує, що модель даних придатна для службового перегляду без додаткових звітів.

Окремо перевірено, що сторінка особистого кабінету не показує скасовані бронювання. Така поведінка відповідає очікуванням клієнта, який хоче бачити актуальні подорожі. Водночас адміністративна частина не втрачає скасований запис, тому історія обліку зберігається.

Тестування інтерфейсу підтвердило, що скріншоти, наведені у звіті, відображають реальні сторінки проєкту. Вони демонструють не макет, а робочий застосунок з даними, формами, повідомленнями й адміністративними списками.

Таблиця 3.2 – Перевірка основних функціональних сценаріїв

Сценарій	Дія	Очікуваний результат	Статус
Каталог турів	Відкрити головну сторінку	Відображено активні тури з основними даними	Виконано
Пошук	Ввести країну, місто або готель	Список турів звужується відповідно до запиту	Виконано
Фільтр	Обрати напрям у списку	Показано тури вибраного напряму	Виконано
Гість	Відкрити сторінку туру без входу	Показано форму заявки, а не бронювання	Виконано

Сценарій	Дія	Очікуваний результат	Статус
Заявка	Надіслати форму гостя	Створено TourRequest і показано повідомлення	Виконано
Реєстрація	Створити акаунт клієнта	Створено User і Client	Виконано
Вхід	Увійти під клієнтом	Доступний особистий кабінет	Виконано
Бронювання	Надіслати BookingForm	Створено Booking зі статусом «Нове»	Виконано
Розрахунок суми	Вказати кількість туристів	total_amount = base_price * tourists_count	Виконано
Призначення менеджера	Створити бронювання	Обрано активного менеджера	Виконано
Кабінет	Перейти до «Мої бронювання»	Показано бронювання поточного клієнта	Виконано
Скасування	Натиснути кнопку скасування	Статус змінено на cancelled	Виконано
Admin	Відкрити список бронювань	Доступні фільтри, пошук, статуси	Виконано
Admin action	Схвалити вибрані бронювання	Статус змінено на confirmed	Виконано
Inline	Додати платіж у картці бронювання	Payment збережено в межах Booking	Виконано

На рисунку 3.7 показано головну сторінку адміністративної панелі. У ній доступні моделі внутрішнього обліку, що відповідають спроєктованим таблицям бази даних.

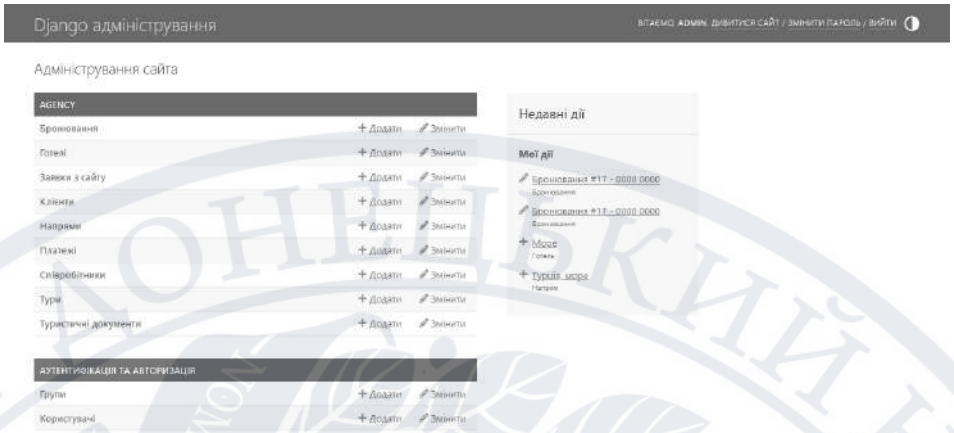

Рисунок 3.7 – Головна сторінка Django Admin

Список бронювань у Django Admin наведено на рисунку 3.8. Він демонструє службовий формат роботи менеджера: таблиця, фільтри, пошук і можливість швидкої зміни статусу.

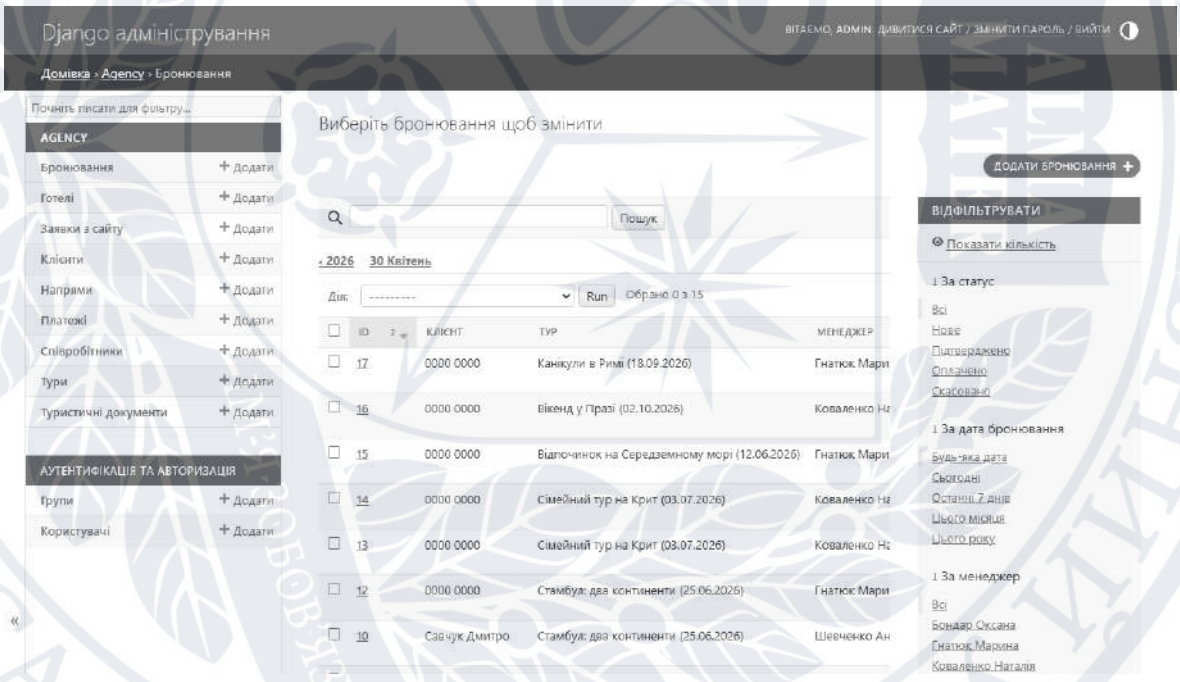

Рисунок 3.8 – Список бронювань в адміністративній панелі

На рисунку 3.9 показано редагування бронювання з вбудованими платежами та документами. Це підтверджує, що фінансовий і документальний супровід пов'язаний з основною операцією обліку.

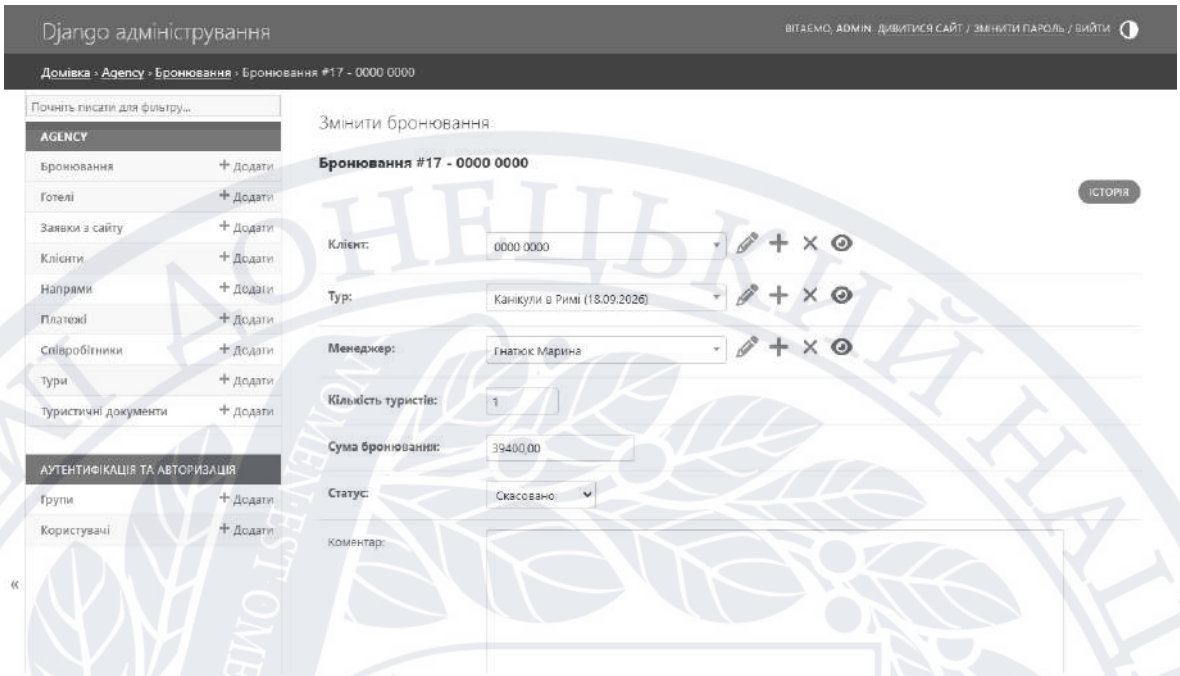

Рисунок 3.9 – Картка бронювання з платежами та документами

3.4. Аналіз продуктивності та оптимізація

Аналіз продуктивності системи виконано на рівні архітектурних рішень і типових запитів. Оскільки дипломний проєкт є навчальним прототипом, навантажувальне тестування з великою кількістю користувачів не проводилося. Проте в коді вже застосовано базові підходи, які зменшують кількість запитів до бази даних і підвищують масштабованість.

Для сторінки каталогу використано `select_related` для завантаження напряму й готелю. Для особистого кабінету використано `select_related` для туру, напряму, готелю та працівника, а також `prefetch_related` для платежів і документів. Це важливо, оскільки без таких оптимізацій кожна картка могла б породжувати додаткові SQL-запити.

На рівні бази даних подальша оптимізація може включати створення індексів для полів, що часто використовуються у фільтрах і сортуванні. Для бронювань це `status`, `booking_date`, `employee_id` і `client_id`; для турів - `active`, `start_date`, `destination_id`; для платежів - `payment_date` і `method`; для заявок - `status` і `created_at`.

Окремої уваги потребує контроль кількості місць. У поточній реалізації поле `seats_total` зберігає загальну кількість місць у турі, але система не блокує створення бронювань понад доступний залишок. Для промислового використання необхідно додати обчислення зайнятих місць за активними бронюваннями, транзакційне резервування та перевірку конкурентних запитів.

Безпекова оптимізація має включати розширення рольової моделі. Нині працівник із доступом до адміністративної панелі може бачити багато сутностей. У реальному агентстві доцільно розділити права менеджера, бухгалтера й директора: менеджер працює з клієнтами та бронюваннями, бухгалтер - з оплатами, керівник - зі звітами та налаштуваннями.

Потрібно також удосконалити роботу з файлами документів. Для демонстраційного проєкту достатньо поля FileField, однак у промисловому середовищі необхідні обмеження типів файлів, перевірка розміру, контроль доступу, резервне копіювання, шифрування або захищене сховище. Це особливо важливо через наявність персональних даних.

У майбутньому система може отримати модуль звітності: кількість бронювань за період, сума оплат, популярні напрями, робота менеджерів, частка скасувань і незавершених заявок. Така аналітика може бути побудована на вже наявних таблицях, оскільки модель зберігає ключові дати, статуси, суми й відповідальних працівників.

Загалом продуктивність і масштабованість розробленої системи відповідають рівню навчального прототипу та можуть бути підвищені без повної зміни архітектури. Обраний стек Django + PostgreSQL дозволяє поступово додавати індекси, кешування, пагінацію, ролі, API та аналітичні запити, зберігаючи основну модель даних стабільною.

Подальша оптимізація може також включати пагінацію на клієнтських сторінках, якщо кількість турів значно зросте. Django Admin уже має власні механізми посторінкового виведення, але публічний каталог у майбутньому також потребуватиме обмеження кількості карток на сторінці.

Для підвищення надійності варто додати автоматизовані тести Django. Вони можуть перевіряти створення клієнта після реєстрації, заборону доступу до чужих бронювань, коректність розрахунку суми, зміну статусу та створення заявки гостем. Такі тести зроблять систему стійкішою до майбутніх змін.

Ще одним напрямом розвитку є створення API. Якщо туристичне агентство захоче мати мобільний застосунок або окремий сучасний фронтенд, поточна модель даних може бути використана як серверна основа. Для цього доцільно додати Django REST Framework і описати ресурси турів, бронювань та заявок.

Таблиця 3.3 – Напрями оптимізації та розвитку системи

Напрям	Поточний стан	Рекомендоване вдосконалення
Індексація	Автоматичні індекси первинних і зовнішніх ключів	Додати індекси для status, booking_date, start_date, payment_date
Залишок місць	Зберігається загальна кількість місць	Реалізувати транзакційне резервування доступних місць
Ролі	Базовий доступ через staff/admin	Розділити права менеджера, бухгалтера й директора
Звіти	Дані зберігаються, але окремих звітів немає	Додати аналітику продажів, оплат, скасувань і заявок
Файли	FileField для документів	Додати перевірку типів, розміру, доступу й резервне копіювання
Тести	Виконано ручну перевірку сценаріїв	Додати автоматизовані Django TestCase
API	Реалізовано серверні HTML-сторінки	Додати REST API для мобільного або окремого фронтенду
Пагінація	Каталог виводить усі активні тури	Додати посторінковий вивід для великої кількості турів

На рисунку 3.10 наведено список турів у службовому інтерфейсі. Він демонструє, як адміністратор може контролювати активність, дати, ціну, напрям і готель кожної пропозиції.

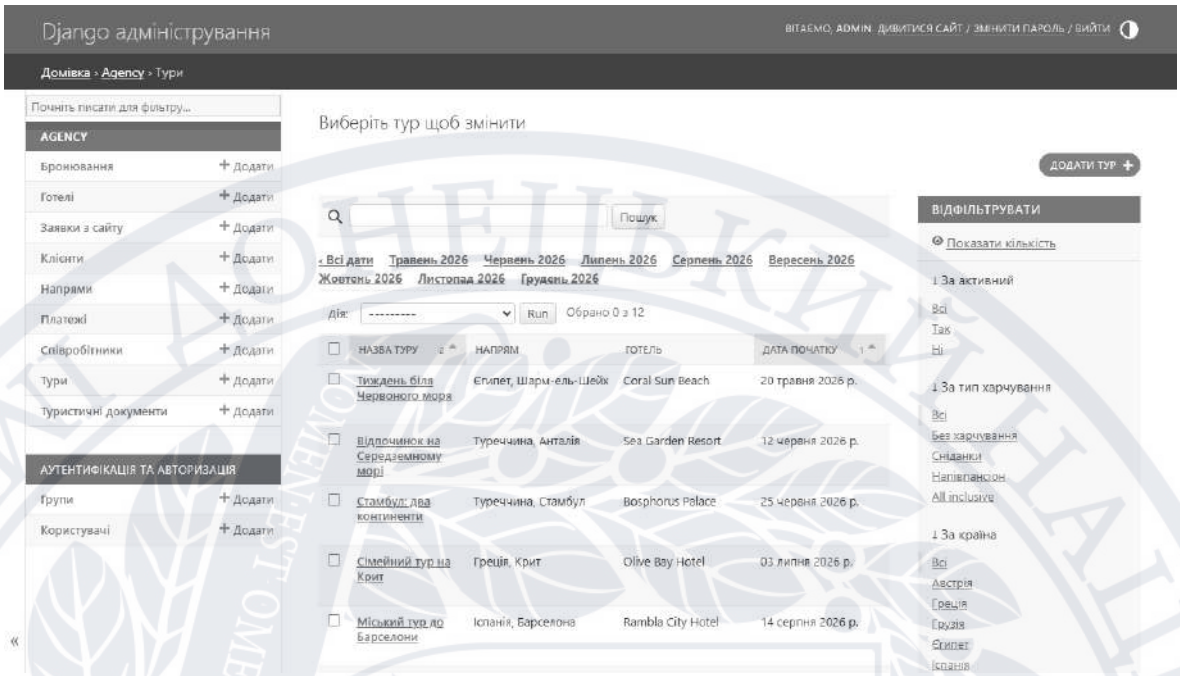

Рисунок 3.10 – Список турів в адміністративній панелі

На рисунку 3.11 подано загальну форму заявки на індивідуальний підбір. Вона дозволяє отримати звернення навіть тоді, коли клієнт ще не обрав конкретний тур.

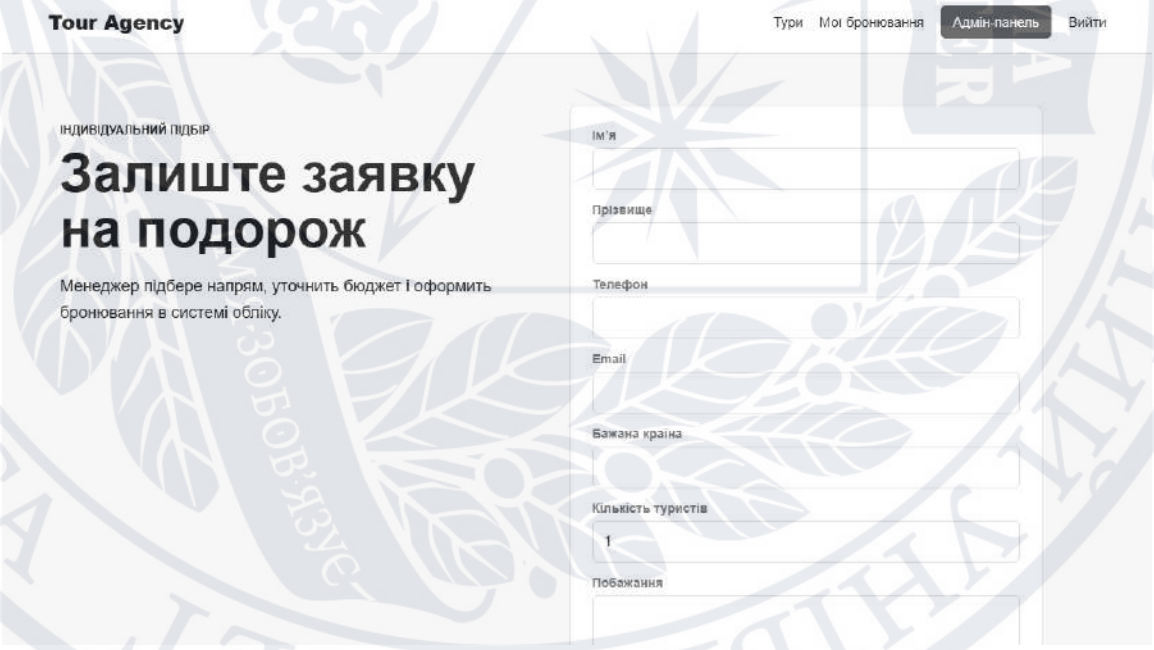

Рисунок 3.11 – Форма заявки на індивідуальний підбір подорожі

Висновки до розділу 3

У третьому розділі описано практичну реалізацію бази даних і веб-застосунку на Django та PostgreSQL, налаштування адміністративного й

клієнтського інтерфейсів, а також перевірку основних сценаріїв роботи. За результатами тестування встановлено, що прототип виконує поставлені функції та має резерв для подальшого розширення, оптимізації й посилення безпеки.

ВИСНОВКИ

У роботі систематизовано предметну область внутрішнього обліку туристичного агентства та визначено, що центральною операційною сутністю є бронювання, яке поєднує клієнта, тур, менеджера, платежі й туристичні документи.

Проаналізовано бізнес-процеси обліку клієнтів, працівників, напрямів, готелів, турів, бронювань, оплат, документів і заявок із сайту. На основі аналізу сформульовано функціональні та нефункціональні вимоги до інформаційної системи.

Побудовано концептуальну модель даних, у якій виділено дев'ять основних сутностей. Логічне проєктування виконано з урахуванням нормалізації, контролю допустимих значень, унікальних обмежень і правил збереження історії операцій.

Фізичну модель реалізовано засобами Django ORM і PostgreSQL. У проєкті використано моделі, міграції, зовнішні ключі, поля з обмеженим набором значень, абстрактну модель міток часу та конфігурацію підключення до PostgreSQL через змінні середовища.

Реалізовано користувацький веб-інтерфейс для перегляду турів, пошуку, фільтрації, реєстрації, входу, створення бронювання, скасування бронювання та надсилання заявки. Для працівників налаштовано Django Admin із пошуком, фільтрами, inline-формами платежів і документів та груповими діями.

Проведено перевірку основних сценаріїв: перегляд каталогу, створення заявки гостем, реєстрація клієнта, створення бронювання, автоматичне призначення менеджера, перегляд особистого кабінету, скасування бронювання і службове опрацювання записів в адміністративній панелі.

Практичне значення роботи полягає в отриманні робочого прототипу, який демонструє зв'язок між спроєктованою базою даних і реальними інтерфейсами користувача. Розроблена система може бути розширена модулями звітності,

детальнішими ролями, контролем залишку місць, API та автоматизованими тестами.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Зелінська О. В., Січко Т. В., Потапова Н. А., Якубич К. О. Методичні рекомендації до виконання, оформлення та захисту кваліфікаційної (бакалаврської) роботи здобувачами спеціальності 122 «Комп'ютерні науки». Вінниця: ДонНУ імені Василя Стуса, 2024. 27 с.
2. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ: ДП «УкрНДНЦ», 2016.
3. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ: ДП «УкрНДНЦ», 2016.
4. ISO/IEC/IEEE 12207:2017. Systems and software engineering. Software life cycle processes. Geneva: ISO, 2017.
5. ISO/IEC/IEEE 29148:2018. Systems and software engineering. Life cycle processes. Requirements engineering. Geneva: ISO, 2018.
6. ISO/IEC 25010:2011. Systems and software engineering. Systems and software Quality Requirements and Evaluation. System and software quality models. Geneva: ISO, 2011.
7. ISO/IEC 27001:2022. Information security, cybersecurity and privacy protection. Information security management systems. Geneva: ISO, 2022.
8. Sommerville I. Software Engineering. 10th ed. Boston: Pearson, 2016.
9. Django Software Foundation. Django documentation. URL: https://docs.djangoproject.com/ (дата звернення: 10.05.2026).
10. Django Software Foundation. Model field reference. URL: https://docs.djangoproject.com/en/stable/ref/models/fields/ (дата звернення: 10.05.2026).
11. Django Software Foundation. QuerySet API reference. URL: https://docs.djangoproject.com/en/stable/ref/models/querysets/ (дата звернення: 10.05.2026).

12. Django Software Foundation. The Django admin site. URL: https://docs.djangoproject.com/en/stable/ref/contrib/admin/ (дата звернення: 10.05.2026).
13. Django Software Foundation. User authentication in Django. URL: https://docs.djangoproject.com/en/stable/topics/auth/ (дата звернення: 10.05.2026).
14. PostgreSQL Global Development Group. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 10.05.2026).
15. PostgreSQL Global Development Group. Data Definition: Constraints. URL: https://www.postgresql.org/docs/current/ddl-constraints.html (дата звернення: 10.05.2026).
16. PostgreSQL Global Development Group. Indexes. URL: https://www.postgresql.org/docs/current/indexes.html (дата звернення: 10.05.2026).
17. Python Software Foundation. Python Documentation. URL: https://docs.python.org/3/ (дата звернення: 10.05.2026).
18. WHATWG. HTML Living Standard. URL: https://html.spec.whatwg.org/ (дата звернення: 10.05.2026).
19. MDN Web Docs. CSS: Cascading Style Sheets. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 10.05.2026).
20. OWASP Foundation. OWASP Application Security Verification Standard. URL: https://owasp.org/www-project-application-security-verification-standard/ (дата звернення: 10.05.2026).
21. OWASP Foundation. OWASP Top 10:2021. URL: https://owasp.org/Top10/ (дата звернення: 10.05.2026).
22. Date C. J. An Introduction to Database Systems. 8th ed. Boston: Pearson, 2003.
23. Elmasri R., Navathe S. B. Fundamentals of Database Systems. 7th ed. Boston: Pearson, 2016.

24. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts. 7th ed. New York: McGraw-Hill, 2019.
25. Connolly T., Begg C. Database Systems: A Practical Approach to Design, Implementation, and Management. 6th ed. Boston: Pearson, 2015.
26. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York: McGraw-Hill, 2020.
27. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2002.
28. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1994.
29. W3C. Web Content Accessibility Guidelines (WCAG) 2.2. URL: https://www.w3.org/TR/WCAG22/ (дата звернення: 10.05.2026).
30. Git Project. Git Documentation. URL: https://git-scm.com/doc (дата звернення: 10.05.2026).
31. Mozilla Developer Network. HTML forms. URL: https://developer.mozilla.org/en-US/docs/Learn/Forms (дата звернення: 10.05.2026).
32. Mozilla Developer Network. HTTP response status codes. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status (дата звернення: 10.05.2026).
33. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: Doctoral dissertation. University of California, Irvine, 2000.
34. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. Boston: Prentice Hall, 2008.
35. Freeman E., Robson E., Bates B., Sierra K. Head First Design Patterns. 2nd ed. Sebastopol: O'Reilly Media, 2020.
36. Larman C. Applying UML and Patterns. 3rd ed. Upper Saddle River: Prentice Hall, 2004.

37. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 2nd ed. Boston: Addison-Wesley, 2005.
38. Nielsen J. Usability Engineering. San Francisco: Morgan Kaufmann, 1993.
39. Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd ed. Berkeley: New Riders, 2014.
40. Kleppmann M. Designing Data-Intensive Applications. Sebastopol: O'Reilly Media, 2017.

ДОДАТОК А

Фрагменти програмної реалізації

У додатку наведено скорочені фрагменти коду, які демонструють принципи реалізації моделі бронювання, форми бронювання та адміністративної дії. Повні файли розміщені у проєкті `tour-agency-django`.

```
class Booking(TimestampedModel):
    client = models.ForeignKey(Client, on_delete=models.PROTECT,
related_name='bookings')
    tour = models.ForeignKey(Tour, on_delete=models.PROTECT,
related_name='bookings')
    employee = models.ForeignKey(Employee, on_delete=models.PROTECT,
related_name='bookings')
    booking_date = models.DateField(auto_now_add=True)
    tourists_count = models.PositiveIntegerField(default=1)
    total_amount = models.DecimalField(max_digits=10, decimal_places=2)
    status = models.CharField(max_length=20, choices=STATUS_CHOICES,
default=STATUS_NEW)

class BookingForm(forms.ModelForm):
    class Meta:
        model = Booking
        fields = ('tourists_count', 'comment')

    def clean_tourists_count(self):
        tourists_count = self.cleaned_data['tourists_count']
        if tourists_count < 1:
            raise forms.ValidationError('Кількість туристів має бути не
менше 1.')
        return tourists_count

@admin.action(description='Схвалити вибрані бронювання')
def approve_bookings(self, request, queryset):
    updated = queryset.exclude(status=Booking.STATUS_CANCELLED).update(
        status=Booking.STATUS_CONFIRMED
    )
    self.message_user(request, f'Схвалено бронювань: {updated}.')
```

ДОДАТОК Б

Додаткові екранні форми

У додатку наведено додатковий екран, що ілюструє форму заявки на індивідуальний підбір подорожі та підтверджує наявність окремого каналу роботи з потенційним клієнтом.

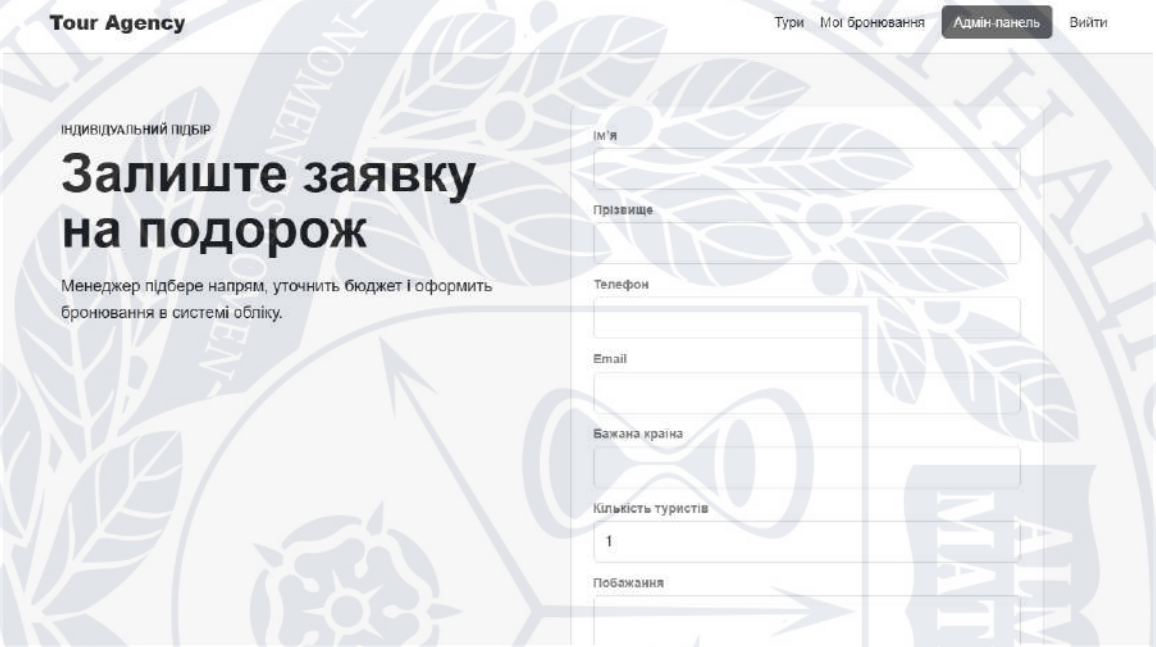

Рисунок Б.1 – Додаткова екранна форма заявки з сайту

ДОДАТОК В

Команди запуску та перевірки проєкту

Наведені команди відображають базовий порядок розгортання демонстраційного середовища, створення структури бази даних, наповнення тестовими даними та запуску локального сервера. Вони доповнюють опис реалізації і можуть бути використані для повторної перевірки роботи застосунку.

```
pip install -r requirements.txt
python manage.py makemigrations
python manage.py migrate
python manage.py seed_demo
python manage.py createsuperuser
python manage.py runserver

http://127.0.0.1:8000/
http://127.0.0.1:
8000/admin/
```