НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
В.о. завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп'ютерні системи та мережі”

спеціальності 123 “Комп’ютера інженерія”

на тему: Веб-застосунок для бронювання квитків

Виконав : студент 4 курсу, групи ІО-14
(шифр групи)

Риндя Ілля Андрійович
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент, Каплунов А. В.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) асистент, Нікольський С. С.
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент старший викладач, Тимофєєва Ю. С.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп'ютерні системи та мережі”

спеціальності 123 “Комп’ютера інженерія”

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____2025 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Ринді Іллі Андрійовича

1. Тема проєкту *Веб-застосунок для бронювання квитків*
керівник проєкту Каплунов Артем Володимирович, асистент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 23 травня 2025 року №1705-с

2. Термін здачі студентом закінченого проєкту *02.06.2025*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Огляд веб-застосунків для бронювання квитків.

Розділ 2. Огляд технологій для розробки веб-застосунку.

Розділ 3. Розробка програмного забезпечення.

Розділ 4. Дослідження та аналіз розробленого веб-застосунку.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (діаграма класів), алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	ас. Нікольський С. С.		

7. Дата видачі завдання 25.01.2025

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*25.01.2025-31.01.2025*	
2.	*Вивчення та аналіз завдання*	*31.01.2025-31.03.2025*	
3.	*Розробка архітектури та загальної структури системи*	*31.03.2025-16.04.2025*	
4.	*Розробка структур окремих підсистем*	*16.04.2025-22.04.2025*	
5.	*Програмна реалізація системи*	*22.04.2025-02.05.2025*	
6.	*Оформлення пояснювальної записки*	*02.05.2025-17.05.2025*	
7.	*Захист програмного продукту*	*16.06.2025*	
8.	*Передзахист*	*02.06.2025*	
9.	*Захист*	*16.06.2025*	

Студент-дипломник _____ Ілля РИНДЯ
(підпис)

Керівник проєкту _____ Артем КАПЛУНОВ
(підпис)

АНОТАЦІЯ

Ця дипломна робота присвячена розробці веб-застосунку для автоматизованого бронювання квитків на різноманітні події. Метою було створення оптимального інтерфейсу, що дозволяє користувачам переглядати події, обирати місця та здійснювати бронювання. Система розроблена з використанням Python та Django, а інтерфейс побудовано за допомогою Django Templates. База даних організована навколо подій, місць та квитків, з реалізованою функціональністю реєстрації, авторизації та адміністрування. Розроблений застосунок є гнучким та адаптованим для різноманітних подій.

Ключові слова: вебзастосунок, бронювання квитків, Django, Python, події, квитки, Django Templates.

ANNOTATION

This thesis focuses on developing a web application for automated ticket booking for various events. The goal was to create an optimal interface enabling users to view events, select seats, and make reservations. The system is built using Python and Django, with the user interface constructed via Django Templates. The database is structured around events, seats, and tickets, incorporating registration, authorization, and administrative functionalities. The developed application is flexible and adaptable for diverse events (performances, concerts, conferences).

Keywords: web application, ticket booking, Django, Python, events, tickets, Django Templates.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Веб-застосунок для	3		
			бронювання квитків			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Веб-застосунок для	65		
			бронювання квитків			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 ДА***	Веб-застосунок для	1		
			бронювання квитків			
			Схема Django моделей			
	А4	***ІАЛЦ.467200.005 ДБ***	Веб-застосунок для	1		
			бронювання квитків			
			Структурна схема MVT			
			патерну в Django			
	А4	***ІАЛЦ.467200.006 ДВ***	Веб-застосунок для	1		
			бронювання квитків			
			Алгоритм дій веб-застосунку			
	А4	***ІАЛЦ.467200.007 ДГ***	Веб-застосунок для	9		
			бронювання квитків			
			Текст програмного коду			

					# *ІАЛЦ.467200.001 ОА*		
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*			
Розроб		Риндя І.А.			*Веб-застосунок для бронювання квитків* Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перев		Каплунов А.В.					
						“КПІ ім. Ігоря Сікорського” ФІОТ ІО-14	

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Веб-застосунок для бронювання квитків*»

Київ – 2025

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ 2

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ 2

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ 2

4. ДЖЕРЕЛА РОЗРОБКИ 2

5. ТЕХНІЧНІ ВИМОГИ 3

5.1 Вимоги до розробленого продукту 3

5.2 Вимоги до програмного забезпечення 3

5.3 Вимоги до апаратної частини 3

6. ЕТАПИ РОЗРОБКИ 3

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання стосується створення та підтримки вебзастосунку для бронювання квитків на різноманітні події.

Сфера застосування охоплює використання застосунку кінцевими користувачами, які бажають переглядати майбутні події, обирати доступні місця та бронювати квитки онлайн у зручний спосіб.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки веб-застосунку є завдання, визначене для виконання кваліфікаційної роботи рівня «бакалавр інженерії програмного забезпечення», яке було затверджено факультетом «Інформатики та обчислювальної техніки» кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є створення функціонального вебзастосунку, який дозволить користувачам переглядати події, обирати місця у залі та здійснювати бронювання квитків у кілька кроків. Призначенням системи є забезпечення простого процесу взаємодії між організаторами подій та глядачами, підвищення зручності планування дозвілля для користувачів.

4 ДЖЕРЕЛА РОЗРОБКИ

До основних джерел інформації, використаних у процесі розробки, належать офіційна документація фреймворку Django, матеріали з відкритих технічних ресурсів, приклади реалізації подібних систем, а також спеціалізована література з вебпрограмування, розробки інтерфейсів і проєктування баз даних.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Простий та інтуїтивно зрозумілий інтерфейс для користувачів.
- Реалізація автентифікації та авторизації користувачів.
- Надати можливість користувачам переглядати доступні події.
- Можливість вибору місць та бронювання квитків на події.

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- Сучасний веб-браузер.
- Visual Studio Code.
- Python версії 3.12

5.3. Вимоги до апаратної частини

- ЦП Intel® Core (TM) i3-2100T або вище.
- ROM не менше ніж 64 ГБ.
- RAM не менше ніж 4 ГБ.
- Доступ до мережі Інтернет

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	25.01.2025-31.01.2025
Вивчення та аналіз завдання	31.01.2025-31.03.2025
Розробка архітектури та загальної структури системи	31.03.2025-16.04.2025
Розробка структур окремих частин системи	16.04.2025-22.04.2025
Програмна реалізація системи	22.04.2025-02.05.2025
Виправлення помилок	02.05.2025-08.05.2025
Оформлення пояснювальної записки	02.05.2025-17.05.2025

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Веб-застосунок для бронювання квитків*»

Київ – 2025

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 3

ВСТУП 4

РОЗДІЛ 1. ОГЛЯД ВЕБ-ЗАСТОСУНКІВ ДЛЯ БРОНЮВАННЯ КВИТКІВ 6

1.1 Актуальність та класифікація систем бронювання квитків на події ... 6

1.2 Аналіз базового функціоналу 7

1.3 Аналіз розширеного функціоналу та інновацій 9

1.4 Огляд популярних платформ для бронювання квитків на події 10

1.5 UI/UX аспекти 11

1.6 Проблеми та тенденції розвитку систем бронювання квитків на події 13

1.6.1 Основні Проблеми 13

1.6.2 Ключові Тенденції Розвитку 14

ВИСНОВОК ДО РОЗДІЛУ 1 16

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБ-ЗАСТОСУНКУ 17

2.1 Вибір та обґрунтування технологічного стеку 17

2.2 Технології Frontend розробки 18

2.2.1 Базові мови Frontend 18

2.2.2 CSS фреймворки, методології та препроцесори 19

2.2.3 JavaScript фреймворки/бібліотеки та обраний підхід до рендерингу 20

2.2.4 Аргументація вибору Django Templates 21

2.3 Технології Backend розробки 22

2.3.1 Огляд популярних мов та платформ для Backend 22

2.3.2 Python 23

2.3.3 Django 24

2.3.4 Архітектурні підходи та API в контексті Django 26

2.4 Системи управління базами даних (СУБД) 26
2.4.1 Реляційні СУБД (SQL) 27
2.4.2 Нереляційні СУБД (NoSQL) 28
2.5 Технології інфраструктури та DevOps 28
2.5.1 Хмарні платформи 28
2.5.2 Контейнеризація 29
2.5.3 CI/CD та GitHub Workflows 29
2.5.4 Моніторинг та логування 31
ВИСНОВОК ДО РОЗДІЛУ 2 32
РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 33
3.1 Опис процесу розробки та застосована методологія 33
3.2 Налаштування середовища та інструменти розробки 34
3.3 Проектування бази даних 35
3.4 Розробка Backend частини 37
3.4.1 Архітектурний патерн Model-View-Template (MVT) в Django 37
3.4.2 Структура Django-проекту та додатків (apps) 38
3.4.3 Реалізація представлень (Views) та бізнес-логіки 39
3.4.4 Налаштування маршрутизації 39
3.4.5 Конфігурація адміністративної панелі Django Admin 39
3.4.6 Інтеграція платіжної системи Stripe 40
3.5 Розробка Frontend частини 42
3.5.1 Структура шаблонів та успадкування 42
3.5.2 Відображення динамічних даних у шаблонах 42
3.5.3 Робота зі статичними файлами 42
3.6 Реалізація ключових алгоритмів та логіки 43
3.7 Аспекти безпеки застосунку 45
3.8 Налаштування CI/CD за допомогою GitHub Workflows 45
ВИСНОВОК ДО РОЗДІЛУ 3 47

РОЗДІЛ 4. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОГО ВЕБ-ЗАСТОСУНКУ 48
4.1 Цілі та напрямки аналізу результатів розробки 48
4.2 Загальний опис реалізованого веб-застосунку та його архітектури .. 48
4.3 Аналіз реалізованого функціоналу 49
4.3.1 Головна сторінка та відображення подій 49
4.3.2 Функціонал пошуку подій 51
4.3.3 Сторінка детальної інформації про подію 52
4.3.4 Процес купівлі квитків та інтеграція з Stripe 53
4.3.5 Система автентифікації та авторизації користувачів 54
4.3.6 Інформаційні сторінки "Про нас" та "Контакти" 55
4.4 Аналіз користувацького інтерфейсу (UI) та досвіду користувача (UX) 56
4.5 Оцінка використаного стеку технологій у практичній реалізації 57
4.6 Відповідність поставленим завданням 58
4.7 Обмеження та напрямки подальшого розвитку 58
ВИСНОВОК ДО РОЗДІЛУ 4 60
ВИСНОВКИ 61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 64

ПЕРЕЛІК СКОРОЧЕНЬ

БД	База даних
HTTP	(HyperText Transfer Protocol) Протокол передачі даних
HTML	(HyperText Markup Language) Мова розмітки гіпертексту
CSS	(Cascading Style Sheets) Каскадні таблиці стилів
URL	(Uniform Resource Locator) Стандартизований вказівник на ресурс
SQL	(Structured Query Language) Мова для взаємодії з базами даних
MVT	(Model View Template) Патерн проектування, який використовується для веб-розробки в рамках фреймворку Django
UI/UX	User Interface/User Experience
SPA	Single-Page Application
CI/CD	(Continuous Integration/Continuous Delivery) Практика в розробці програмного забезпечення, яка автоматизує процеси побудови, тестування та випуску програмного продукту
ORM	(Object-Relational Mapping) Технологія в програмуванні, яка дозволяє працювати з реляційними базами даних, використовуючи об'єкти, як у мові програмування
API	(Application Programming Interface) Інтерфейс програмування додатків, який визначає правила, за допомогою яких різні програми або системи взаємодіють між собою, обмінюються даними та надають доступ до своїх функцій

ВСТУП

Культурні, розважальні та спортивні заходи – невіддільна частина життя багатьох із нас. Концерти улюблених гуртів, театральні прем'єри чи напружені спортивні матчі дарують яскраві враження та емоції. Проте, шлях до цих емоцій часто затьмарюється незручностями: доводиться витрачати час на поїздки до кас, стояти в чергах, а іноді й стикатися з браком інформації про вільні місця, якщо купувати квитки по-старому. У сучасному динамічному світі, де кожна хвилина на рахунку, виникає закономірне бажання мати зручніші способи придбання квитків.

Стрімкий розвиток вебтехнологій відкрив нам доступ до онлайн-сервісів, які значно полегшують отримання різноманітних послуг. Вебзастосунки стали нашими щоденними помічниками: через них ми купуємо товари, знаходимо інформацію та бронюємо послуги, не виходячи з дому чи перебуваючи в дорозі. Тому створення спеціалізованого вебзастосунку для бронювання квитків – це актуальне завдання. Такий сервіс допоможе вирішити проблеми традиційних методів купівлі та задовольнить потребу людей у швидкому, простому та прозорому доступі до квитків.

Метою цієї бакалаврської роботи є створення вебзастосунку, який стане для користувачів ефективним і зручним інструментом пошуку, вибору та купівлі квитків на найрізноманітніші події – чи то концерти, театральні вистави, чи спортивні змагання. Передбачається, що застосунок забезпечить повний цикл взаємодії: від легкого пошуку потрібного заходу та детального ознайомлення з інформацією про нього, до вибору вільних місць на інтерактивній схемі, безпечної онлайн-оплати та отримання електронного квитка.

Втілення цього проєкту дозволить запропонувати сучасну платформу. Вона не лише спростить процес придбання квитків для звичайних користувачів,

а й допоможе організаторам заходів оптимізувати свою роботу завдяки автоматизації продажів, обліку та комунікації з аудиторією.

РОЗДІЛ 1. ОГЛЯД ВЕБ-ЗАСТОСУНКІВ ДЛЯ БРОНЮВАННЯ КВИТКІВ

1.1 Актуальність та класифікація систем бронювання квитків на події

Актуальність систем бронювання квитків зумовлена очевидними перевагами для всіх учасників процесу. Для користувачів це, перш за все, зручність: можливість у будь-який час та з будь-якого місця ознайомитись з повною афішею подій, порівняти ціни та умови, обрати найкращі місця та миттєво придбати квиток, уникаючи черг та необхідності відвідувати фізичні каси. Для організаторів подій веб-платформи надають значно ширше охоплення аудиторії, ефективні інструменти для маркетингу та просування, можливість автоматизувати процеси продажу та обліку, збирати цінні дані про свою аудиторію та оптимізувати логістику заходу, знижуючи операційні витрати.

Зважаючи на різноманіття заходів, системи бронювання квитків можна класифікувати за кількома критеріями, хоча чіткі межі часто розмиваються, особливо у випадку універсальних платформ. За широтою охоплення подій розрізняють спеціалізовані платформи, що фокусуються на конкретному типі подій (наприклад, лише кінотеатри), широкопрофільні агрегатори, які прагнуть охопити максимально можливий спектр подій у певному регіоні чи країні (концерти, спорт, театр, фестивалі), та глобальні платформи, що діють на міжнародному рівні.

За типом подій системи можуть бути орієнтовані на культурно-розважальні, спортивні, кінопокази, ділові та освітні заходи, а також дозвілля та атракціони.

Щодо моделі роботи, виділяють первинний ринок, де діють офіційні продавці за прямими угодами з організаторами (як-от сайти-агрегатори чи

власні сайти майданчиків); вторинний ринок, що включає платформи для перепродажу квитків між користувачами, часто пов'язаний з проблемою спекуляції; та платформи для організаторів (SaaS), які надають інструменти безпосередньо організаторам для створення власних сторінок продажу.

Даний дипломний проєкт спрямований на розробку універсального веб-застосунку, здатного обслуговувати широкий спектр типів подій на первинному ринку, надаючи гнучкі інструменти як для покупців, так і для організаторів. Аналіз існуючих рішень проводитиметься з урахуванням вимог до такої універсальної системи.

1.2 Аналіз базового функціоналу

Незалежно від типу події та масштабу платформи, існує ядро функціональних можливостей, необхідних для забезпечення процесу онлайн-бронювання, що забезпечує зручність для користувачів та ефективність для організаторів.

Пошук та фільтрація подій є основою взаємодії, дозволяючи користувачам знаходити заходи за ключовими словами (назва, артист, місце), категоріями, датами та локаціями. Розширена фільтрація дозволяє звузити вибірку за ціною, доступністю місць, рейтингом, тривалістю чи мовою. Каталог подій пропонує структуроване представлення доступних заходів у вигляді списку або сітки карток з ключовою інформацією, а також можливості сортування та різні режими перегляду (календар, карта).

Детальна інформація про подію надається на окремій сторінці, що включає повний опис заходу, програму, біографії учасників, деталі місця проведення (адреса, карта, фото, правила), вікові обмеження, контакти організатора, мультимедійні матеріали та відгуки.

Вибір місць/квитків може бути реалізований по-різному: для фіксованих місць це інтерактивна карта залу з візуальним відображенням доступних, зайнятих та обраних місць; для загального входу – вибір кількості квитків без прив'язки до конкретного місця, з можливістю різних категорій (наприклад, стандарт, ранній вхід); або бронювання за часовими слотами для музеїв чи атракціонів.

Кошик замовлення слугує проміжним етапом, де користувач може переглянути обрані квитки з детальною інформацією та загальною сумою. Критично важливим є механізм тимчасового резервування квитків у кошику з візуальним таймером зворотного відліку.

Реєстрація та Авторизація дозволяють створювати обліковий запис для зберігання персональних даних, історії замовлень, управління підписками та отримання персоналізованих пропозицій, з можливістю "гостьового замовлення". Оформлення замовлення передбачає максимально просту та зрозумілу форму введення даних покупця.

Оплата вимагає інтеграції з надійними платіжними системами, що підтримують банківські картки та бажано Google Pay, Apple Pay, PayPal, з процесом оплати на захищеній HTTPS-сторінці, що відповідає стандартам безпеки. Після успішної оплати слідує підтвердження та доставка квитка — користувач отримує підтвердження на екрані та детальний лист на email, включаючи електронний квиток у форматі PDF з унікальним QR-кодом або штрих-кодом, доступний також в особистому кабінеті.

Особистий кабінет користувача є захищеною зоною для перегляду історії замовлень, завантаження квитків, управління персональними даними, налаштування сповіщень та ініціювання повернень/переоформлень квитків.

Цей набір функцій становить необхідний мінімум для будь-якої сучасної та конкурентоспроможної універсальної платформи бронювання квитків на події.

1.3 Аналіз розширеного функціоналу та інновацій

Щоб виділитися на насиченому ринку та надати користувачам і організаторам додаткову цінність, сучасні платформи впроваджують більш складні та інноваційні функції.

Одним з ключових напрямків є просунута персоналізація. Це включає використання алгоритмів машинного навчання для глибокого аналізу поведінки користувача, що дозволяє формувати високорелевантну стрічку рекомендацій та надсилати персоналізовані сповіщення про події, які відповідають інтересам конкретного користувача. Використовуються підходи колаборативної та контентної фільтрації для досягнення максимальної точності.

Покращені інтерактивні карти залів виходять за межі простого вибору місця, пропонуючи 3D-візуалізацію, фотографії або панорамний перегляд з обраного місця, що допомагає користувачеві краще оцінити видимість та комфорт. Для подій з високим попитом, де квитки швидко розкуповуються, функція списків очікування та сповіщень дозволяє користувачам зареєструвати свій інтерес та автоматично отримувати сповіщення про появу вільних квитків або старт продажів.

Соціальні та ком'юніті функції забезпечують інтеграцію з соціальними мережами для легкого поширення інформації про подію або плани її відвідати. Це включає відображення друзів, які також купили квитки або цікавляться подією, а також розвинені системи користувацьких відгуків та рейтингів подій і майданчиків з можливістю додавання фото, коментування та оцінки корисності відгуків, а також створення секцій запитань і відповідей для кожної події.

Програми лояльності та партнерства передбачають впровадження багаторівневих систем з накопиченням бонусів, ексклюзивними знижками та раннім доступом до продажів для постійних клієнтів. Важливими є й

інструменти для організаторів, що надають їм зручний інтерфейс для самостійного створення та редагування сторінок подій, завантаження схем залів, управління квотами квитків та цінами, налаштування промокодів та відстеження статистики продажів у реальному часі.

Додатково, платформи можуть пропонувати продаж супутніх товарів та послуг, таких як офіційний мерчендайз або страхування квитка, а також забезпечувати мультимовність та мультивалютність для підтримки різних мов інтерфейсу та проведення оплати в різних валютах, що особливо актуально для міжнародних платформ. Впровадження цих функцій значно підвищує конкурентоспроможність платформи та задоволеність як кінцевих користувачів, так і організаторів подій.

1.4 Огляд популярних платформ для бронювання квитків на події

Concert.ua / Karabas.com є лідерами українського ринку, охоплюючи широкий спектр подій, від концертів до дитячих заходів. Їхні сильні сторони — велика локальна афіша, знання місцевої специфіки, зручні інтерфейси та інтерактивні карти українських майданчиків, а також статус офіційних квиткових операторів. До слабких сторін належать орієнтованість переважно на Україну, високі сервісні збори та менша гнучкість функціоналу для організаторів порівняно з глобальними SaaS-рішеннями.

Ticketmaster — найбільший світовий квитковий оператор, що діє в десятках країн, охоплюючи величезну кількість подій. Його переваги — глобальне покриття, ексклюзивні контракти, передові технології боротьби з ботами, потужна інфраструктура та досвід роботи з подіями будь-якого масштабу. Однак, компанія часто критикується за високі сервісні збори,

монопольне становище та непрозоре ціноутворення, а також складні процедури повернення.

Eventbrite фокусується на наданні інструментів для організаторів різноманітних подій. Сильні сторони цієї платформи — гнучкі інструменти для організаторів (створення подій, різні типи квитків, промокоди, аналітика), придатність для нішевих та невеликих заходів, а також інтеграція з соціальними мережами. Як агрегатор для користувачів, Eventbrite може мати менш повну афішу великих комерційних подій порівняно з Ticketmaster чи локальними лідерами.

Прикладом спеціалізованої платформи є сайти мереж кінотеатрів (наприклад, Multiplex в Україні). Їхні сильні сторони — прямий продаж без зайвих комісій, повний та актуальний розклад сеансів саме цієї мережі, власні програми лояльності та зручний вибір місць у знайомих залах. Основна слабка сторона — обмеженість пропозиції лише однією мережею кінотеатрів.

Цей огляд демонструє, що універсальна платформа має поєднувати широту охоплення подій, гнучкість для організаторів та зручність спеціалізованих рішень, пропонуючи при цьому конкурентоспроможні умови та якісний сервіс.

1.5 UI/UX аспекти

В умовах високої конкуренції якість користувацького інтерфейсу та загального досвіду взаємодії часто стає вирішальним фактором у виборі платформи для бронювання квитків. Користувачі очікують не просто функціонального, а інтуїтивно зрозумілого, швидкого та приємного процесу.

До основних вимог до UI/UX універсальної системи бронювання належать: простота та інтуїтивність, що забезпечує логічну та передбачувану навігацію, дозволяючи користувачеві легко знаходити події, розуміти процес

вибору квитків та завершувати покупку з мінімальними зусиллями. Важливою є чітка візуальна ієрархія елементів.

Швидкодія та продуктивність є критично важливими, оскільки час завантаження сторінок, особливо з великою кількістю подій та інтерактивними картами, має бути мінімальним. Для цього використовуються техніки оптимізації зображень, кешування, асинхронного завантаження даних, розділення коду та лінива загрузка.

Адаптивність та кросплатформеність вимагають, щоб застосунок коректно працював та виглядав на всіх типах пристроїв — від настільних комп'ютерів до смартфонів. Дизайн має бути адаптивним або використовуватись окрема мобільна версія/нативний додаток, причому підхід "mobile-first" часто є пріоритетним.

Візуальна привабливість та консистентність забезпечуються сучасним, чистим та естетично приємним дизайном, що підвищує довіру користувачів та комфорт взаємодії. Важливо використовувати якісні зображення та відео, а дизайн має бути консистентним у всіх розділах застосунку.

Чіткий зворотний зв'язок передбачає постійне інформування користувача про поточний стан системи, включаючи показ індикаторів завантаження, повідомлення про успішні дії, таймери резервування та повідомлення про помилки.

Доступність є критично важливим аспектом, що забезпечує можливість користування застосунком для людей з обмеженими можливостями. Це включає дотримання стандартів WCAG: достатня контрастність кольорів, навігація за допомогою клавіатури, текстові альтернативи для зображень, використання семантичної розмітки та ARIA-атрибутів.

Нарешті, ефективність процесу оформлення вимагає мінімізації кроків та полів для заповнення, логічної послідовності дій, використання автозаповнення для зареєстрованих користувачів, а також чітких підказок та валідації введених даних у реальному часі.

Інвестиції в UI/UX безпосередньо впливають на конверсію, рівень задоволеності клієнтів та їх лояльність до платформи.

1.6 Проблеми та тенденції розвитку систем бронювання квитків на події

Незважаючи на зрілість ринку, індустрія онлайн-бронювання квитків на події постійно стикається з викликами та адаптується до нових технологічних і соціальних трендів:

1.6.1 Основні Проблеми

Однією з найгостріших проблем є масштабна спекуляція та боти, особливо для популярних подій. Автоматизовані скрипти масово скуповують квитки в момент старту продажів, позбавляючи реальних фанатів можливості купити їх за номінальною ціною. Згодом ці квитки з'являються на вторинному ринку за значно завищеними цінами. Платформи вимушені впроваджувати складні механізми протидії, такі як CAPTCHA, аналіз поведінки для виявлення ботів, ліміти на кількість квитків, системи попередньої реєстрації та верифікації фанатів, віртуальні черги та динамічні QR-коди.

Серйозну загрозу становить шахрайство та підроблені квитки. Це включає створення фішингових сайтів, що імітують офіційні платформи, а також продаж дублікатів або повністю підроблених квитків, особливо на нерегульованому вторинному ринку. Це вимагає постійного вдосконалення заходів безпеки, ретельної верифікації та інформування користувачів про можливі ризики.

Управління піковими навантаженнями є ще одним значним викликом. Одночасний доступ десятків або сотень тисяч користувачів під час старту продажів квитків на топ-події може призвести до відмови системи. Це вимагає

ретельного планування інфраструктури, використання хмарних технологій для автоматичного масштабування та впровадження систем віртуальної черги для згладжування піків навантаження.

Забезпечення контролю доступу на заході вимагає швидкої та надійної перевірки великої кількості квитків на вході. Це вимагає ефективних рішень для сканування та безшовної інтеграції системи продажу квитків із системою контролю доступу.

Нарешті, конфіденційність даних є фундаментальним аспектом. Збір та обробка персональних даних користувачів вимагає суворого дотримання законодавства про захист даних, забезпечення прозорості політик конфіденційності, отримання згоди користувачів та надійного зберігання даних.

1.6.2 Ключові Тенденції Розвитку

Індустрія бронювання квитків постійно еволюціонує, впроваджуючи новітні технології та підходи для задоволення зростаючих вимог користувачів і організаторів. Розуміння цих тенденцій є ключовим для розробки та підтримки успішного веб-застосунку в довгостроковій перспективі.

Одним із провідних напрямків є поглиблення персоналізації за допомогою AI/ML. Алгоритми штучного інтелекту та машинного навчання стають все досконалішими, дозволяючи створювати унікальний досвід для кожного користувача: від індивідуальних рекомендацій подій до персоналізованого контенту та ціноутворення, базуючись на глибокому аналізі поведінки.

Очевидним пріоритетом стає мобільний пріоритет. Розробка все більше орієнтується на смартфони, які є основним пристроєм для пошуку та купівлі квитків. Це включає створення нативних мобільних додатків, використання мобільних квитків та безконтактний вхід за допомогою телефону.

Технологія блокчейн та NFT-квитки розглядається як потенційне рішення для боротьби зі спекуляцією та підробками. Створення унікальних, незамінних токенів для кожного квитка може забезпечити прозорість володіння, контрольований перепродаж та додаткову цінність.

Гібридні та віртуальні події набувають все більшої популярності, особливо після пандемії COVID-19. Це прискорило розвиток формату онлайн-трансляцій та гібридних подій, що поєднують офлайн та онлайн аудиторію. Платформи бронювання адаптуються, пропонуючи квитки на віртуальний доступ та інтегруючись зі стрімінговими сервісами.

Динамічне ціноутворення стає все більш поширеним, подібно до авіакомпаній та готелів. Квиткові оператори експериментують з цінами, які змінюються залежно від попиту, часу до події та інших факторів, що є інструментом максимізації доходу, хоча й викликає дискусії щодо справедливості.

Платформа бронювання також розвивається в напрямку інтеграції з екосистемою події, стаючи частиною загального досвіду. Це включає інтеграцію з навігацією на майданчику, можливість замовлення їжі або напоїв через додаток та інтерактивні функції під час заходу.

Зростає й увага до сталого розвитку. Платформи можуть надавати інформацію про вуглецевий слід подій, пропонувати опції для його компенсації та заохочувати використання громадського транспорту.

Розуміння цих актуальних тенденцій та адаптація до них є ключовими для розробки та підтримки успішного веб-застосунку в довгостроковій перспективі.

ВИСНОВОК ДО РОЗДІЛУ 1

Цей розділ надав комплексний огляд сучасних вебзастосунків для бронювання квитків, окресливши їхні ключові характеристики, функціональні вимоги та технологічні підходи. Було підкреслено, що успішна універсальна платформа повинна забезпечувати безшовний та інтуїтивно зрозумілий процес пошуку, вибору та купівлі квитків для різноманітних заходів.

Аналіз показав критичну важливість не лише базового функціоналу, а й розширених можливостей, таких як глибока персоналізація на основі AI/ML, розвинені соціальні функції та програми лояльності, а також зручні інструменти для організаторів подій. Визначальними факторами конкурентоспроможності є якість користувацького інтерфейсу, висока продуктивність та робота на мобільних пристроях.

З технологічної точки зору, створення універсальної платформи вимагає використання сучасного стеку, гнучкої та модульної архітектури, а також надійних інтеграцій з платіжними системами та потужної хмарної інфраструктури. Галузь стикається з серйозними викликами, такими як спекуляція квитками та шахрайство, що вимагає постійного вдосконалення заходів безпеки та адаптації до ключових тенденцій розвитку, включаючи боротьбу з ботами, гіперперсоналізацію та розвиток гібридних подій.

Результати цього огляду формують міцне підґрунтя для подальшої розробки, підкреслюючи необхідність проєктування гнучкої архітектури та приділення особливої уваги інтуїтивному UI/UX, персоналізації, продуктивності, масштабованості та сучасним підходам до безпеки.

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБ-ЗАСТОСУНКУ

2.1 Вибір та обґрунтування технологічного стеку

Після всебічного аналізу предметної області та формування вимог до універсального веб-застосунку для бронювання квитків у Розділі 1, критично важливим етапом є вибір технологічного стеку. Сучасний світ веб-розробки пропонує надзвичайно широкий спектр інструментів, бібліотек, фреймворків та платформ. Обґрунтований вибір цих компонентів безпосередньо впливає на швидкість розробки, продуктивність, масштабованість, безпеку, надійність та вартість підтримки майбутнього застосунку. Недооцінка цього етапу може призвести до архітектурних обмежень, що ускладнять подальший розвиток та адаптацію системи до нових вимог.

Метою даного розділу є проведення глибокого огляду ключових технологій, що застосовуються в сучасній веб-інженерії, та надання детального обґрунтування вибору конкретного набору інструментів для розробки універсальної системи бронювання квитків на події в рамках цього дипломного проекту. Особлива увага буде приділена перевагам та недолікам різних підходів, а також специфіці обраних технологій: Python як основної мови програмування, Django як веб-фреймворку, Django Templates для серверного рендерингу інтерфейсу, PostgreSQL як основної системи управління базами даних, Docker для контейнеризації та забезпечення послідовності середовищ розробки й розгортання, а також GitHub Workflows для автоматизації процесів інтеграції та розгортання (CI/CD). Розгляд буде структуровано за ключовими шарами розробки: frontend, backend, управління даними та DevOps.

2.2 Технології Frontend розробки

Frontend є клієнтською частиною веб-застосунку, з якою користувачі безпосередньо взаємодіють. Його завдання - забезпечити інтуїтивно зрозумілий, привабливий, швидкий та адаптивний користувацький інтерфейс на широкому спектрі пристроїв.

2.2.1 Базові мови Frontend

Основою будь-якої веб-сторінки є тріо фундаментальних технологій:

HTML5 (HyperText Markup Language 5): Остання версія стандарту для структурування вмісту веб-сторінок. HTML5 запровадив низку семантичних тегів (таких як <article>, <section>, <nav>, <aside>, <footer>, <header>), які покращують структуру документа, доступність для допоміжних технологій та оптимізацію для пошукових систем (SEO). Також HTML5 включає API для роботи з мультимедіа, графікою, геолокацією, локальним сховище та іншими функціями браузера.

CSS3 (Cascading Style Sheets 3): Мова для опису зовнішнього вигляду та візуального оформлення HTML-документів. CSS3 є модульною і включає розширені можливості для селекторів, адаптивного дизайну, створення складних макетів, анімацій, переходів, тіней, градієнтів та використання власних шрифтів і кастомних властивостей. Для універсальної платформи бронювання, яка має коректно відображатися на різних екранах, ефективне використання CSS3 є критичним.

JavaScript: Високорівнева, динамічно типізована мова програмування, що є стандартом для створення інтерактивності на веб-сторінках. Сучасні версії стандарту ECMAScript значно розширили можливості мови, додавши синтаксичний цукор та нові конструкції, такі як let та const для оголошення

змінних, стрілкові функції, класи, модулі, деструктуризацію, шаблонні рядки, Promises та async/await для роботи з асинхронними операціями. JavaScript використовується для маніпуляції DOM, обробки подій користувача, валідації форм, виконання AJAX-запитів для асинхронного оновлення даних та реалізації складної клієнтської логіки.

TypeScript: Надмножина JavaScript, розроблена Microsoft, що додає опціональну статичну типізацію та інші можливості об'єктно-орієнтованого програмування. Використання TypeScript значно покращує процес розробки великих та складних проектів: статична типізація дозволяє виявляти багато помилок на етапі компіляції, а не під час виконання; покращується автодоповнення коду в IDE та рефакторинг; код стає більш читабельним та зрозумілим, особливо при командній розробці.

2.2.2 CSS фреймворки, методології та препроцесори

Для прискорення розробки, забезпечення консистентності дизайну та кращої організації стилів використовуються різноманітні інструменти:

Bootstrap: Один з найпопулярніших frontend-фреймворків, що надає готову систему сіток, велику кількість перевикористовуваних UI-компонентів (кнопок, форм, навігаційних панелей, модальних вікон) та JavaScript-плагінів. Значно прискорює створення адаптивних інтерфейсів.

Tailwind CSS: CSS-фреймворк, що надає набір низькорівневих утилітарних класів, які можна комбінувати безпосередньо в HTML-розмітці. Це дозволяє створювати повністю кастомні дизайни без написання власного CSS. Перевагою є потенційно менший розмір фінального CSS-файлу, оскільки включаються лише використані утиліти.

CSS-in-JS: Техніки, що дозволяють писати CSS-стилі безпосередньо в JavaScript-коді компонентів. Це покращує інкапсуляцію стилів, дозволяє

використовувати динамічні стилі на основі пропсів компонента та уникати конфліктів імен класів.

Препроцесори CSS (Sass, Less): Розширюють стандартні можливості CSS, додаючи функціонал, властивий мовам програмування: змінні (для зберігання кольорів, шрифтів, розмірів), міксини (перевикористовувані набори стилів), функції, вкладеність селекторів, операції над кольорами та числами, а також можливість розбиття стилів на окремі файли. Це робить CSS-код більш структурованим, модульним та легким для підтримки у великих проектах.

Методології: Набір правил та угод щодо іменування CSS-класів з метою створення передбачуваної, масштабованої та перевикористовуваної структури стилів. BEM(Block-Element-Modificator) пропонує іменувати класи за принципом блок__елемент--модифікатор, що допомагає уникнути конфліктів та полегшує розуміння зв'язків між елементами інтерфейсу.

2.2.3 JavaScript фреймворки/бібліотеки та обраний підхід до рендерингу

Для створення сучасних, високоінтерактивних веб-застосунків (Single Page Applications - SPA) широко використовуються JavaScript-фреймворки та бібліотеки:

React: JavaScript-бібліотека, розроблена кмопанією Meta, для створення користувацьких інтерфейсів. Базується на компонентному підході, де інтерфейс розбивається на незалежні, перевикористовувані компоненти. Використовує Virtual DOM для оптимізації оновлень реального DOM та JSX (розширення синтаксису JavaScript) для декларативного опису UI. Має величезну екосистему, що включає інструменти для розробки, маршрутизації, управління станом та серверного рендерингу статичних сайтів.

Angular: Комплексний frontend-фреймворк, розроблений компанією Google, написаний на TypeScript. Надає повний набір інструментів для

створення великих корпоративних застосунків, включаючи модульну архітектуру, компоненти, сервіси, dependency injection, потужну систему маршрутизації, HTTP-клієнт та інструменти для реактивного програмування.

Vue.js: Прогресивний JavaScript-фреймворк, відомий своєю простотою інтеграції, гнучкістю та пологою кривою навчання. Дозволяє використовувати однофайлові компоненти, що поєднують HTML-шаблон, JavaScript-логіку та CSS-стилі. Має власну екосистему для маршрутизації, управління станом та SSR/SSG.

Незважаючи на потужність SPA-фреймворків, для даного дипломного проекту, де акцент робиться на швидкій реалізації серверної логіки універсальної системи бронювання та враховуючи вибір Python/Django для бекенду, було обрано підхід серверного рендерингу за допомогою вбудованого шаблонного рушія Django Templates. Це потужна, але проста у використанні система, що дозволяє генерувати HTML-сторінки на сервері, вставляючи динамічні дані з backend-логіки.

2.2.4 Аргументація вибору Django Templates

Для рендерингу інтерфейсу обрано Django Templates, що забезпечує тісну інтеграцію та швидкість розробки. Це дозволяє легко отримувати дані з представлень і значно прискорює створення базового функціоналу, такого як відображення списків подій, сторінок деталей та форм, що є критично важливим в умовах обмежених часових ресурсів дипломного проєкту.

Використання Django Templates значно спрощує роботу для Python/Django розробника, дозволяючи зосередитися на серверній логіці без необхідності глибоких знань складних JavaScript-фреймворків для реалізації основного користувацького інтерфейсу. Система успадкування та включення шаблонів надає гнучкість для різних типів подій, дозволяючи створювати базові макети та перевикористовувані компоненти. Ці компоненти можна адаптувати

для відображення специфічної інформації для різних категорій подій; наприклад, базовий шаблон події може бути розширений для відображення карти місць для театральної вистави або програми для конференції.

Хоча рендеринг відбувається на сервері, Django Templates не виключають використання клієнтського JavaScript для покращення користувацького досвіду (UX). Це може включати AJAX-запити для динамічного підвантаження даних без перезавантаження сторінки, клієнтську валідацію форм, інтерактивні елементи інтерфейсу, такі як слайдери та календарі, інтеграцію з бібліотеками для відображення інтерактивних карт місць, а також використання легких JavaScript-бібліотек, наприклад, Alpine.js або HTMX, для додавання реактивності без розробки повноцінного Single Page Application (SPA). Такий підхід дозволяє створити функціональний та достатньо інтерактивний інтерфейс для універсальної системи бронювання, оптимізуючи зусилля на розробку в рамках дипломного проєкту.

2.3 Технології Backend розробки

Backend є критично важливим компонентом, що відповідає за всю бізнес-логіку, управління даними, безпеку та взаємодію з клієнтом та сторонніми сервісами.

2.3.1 Огляд популярних мов та платформ для Backend

Node.js: Платформа, що дозволяє виконувати JavaScript на сервері за допомогою рушія V8. Відома своєю асинхронною, неблокуючою моделлю вводу та виводу, що робить її ефективною для I/O-інтенсивних завдань та додатків реального часу.

Java: Потужна, об'єктно-орієнтована мова, що працює на віртуальній машині Java (JVM). Широко використовується в корпоративному секторі для створення надійних, масштабованих та високопродуктивних систем. Spring Framework надає комплексну інфраструктуру для розробки, включаючи підтримку Dependency Injection, Spring MVC, Spring Data, Spring Security.

PHP: Одна з найстаріших та найпоширеніших мов для веб-розробки. Сучасні фреймворки, такі як Laravel та Symfony, значно покращили якість та продуктивність PHP-розробки, надаючи інструменти для побудови складних застосунків за MVC-патерном та використовуючи менеджер залежностей Composer.

Go: Компільована мова, розроблена Google, з акцентом на простоту, продуктивність та вбудовану підтримку конкурентності. Добре підходить для розробки мікросервісів та високонавантажених систем. Компілюється в єдиний бінарний файл, що спрощує розгортання.

Ruby: Динамічна, об'єктно-орієнтована мова, відома своєю елегантністю. Ruby on Rails є популярним фреймворком, що дотримується принципів "Convention over Configuration" та "Don't Repeat Yourself", що сприяє високій продуктивності розробників. Використовує ActiveRecord для ORM.

2.3.2 Python

Для розробки бекенд-частини цього проєкту обрано мову програмування Python. Цей вибір обумовлений низкою ключових особливостей, що роблять Python ідеальним для створення надійних, масштабованих і гнучких веб-застосунків.

Python є інтерпретованою мовою з динамічною типізацією, що спрощує процес розробки та тестування. Водночас можливість використання типових підказок покращує читабельність коду та підтримує статичний аналіз, роблячи його більш гнучким і надійним. Мова підтримує багатопарадигмове

програмування, включаючи об'єктно-орієнтований, процедурний та функціональний стилі.

Однією з головних переваг Python є його простий синтаксис та висока читабельність, що дозволяє швидко вивчати мову та легко розуміти і підтримувати наявний код. Це забезпечує ефективну командну роботу та знижує поріг входу для нових розробників.

Величезна екосистема та бібліотеки є ще одним вагомим аргументом. Окрім потужної стандартної бібліотеки, Python має доступ до Python Package Index (PyPI), що містить сотні тисяч пакетів для різноманітних завдань. Це включає спеціалізовані бібліотеки для веб-розробки, роботи з даними, наукових обчислень, машинного навчання, виконання HTTP-запитів, а також засоби для ORM.

Python має одну з найбільших та найактивніших спільнот розробників у світі, що гарантує доступ до великої кількості документації, навчальних ресурсів, форумів та конференцій. Це забезпечує постійну підтримку та розвиток екосистеми.

Застосування Python у таких технологічних гігантах, як Google, Instagram, Spotify, Netflix та Dropbox, підтверджує його надійність, масштабованість та придатність для складних проєктів. Здатність до швидкого прототипування та прискореної розробки дозволяє швидко перевіряти ідеї та створювати робочі версії застосунків.

2.3.3 Django

Основним веб-фреймворком для Python обрано Django. Це повнофункціональний фреймворк, що надає комплексний набір інструментів для швидкої та ефективної розробки веб-застосунків, слідуючи архітектурному шаблону MVT (Model-View-Template).

Модель представляє дані застосунку за допомогою Django ORM (Object-Relational Mapper). ORM дозволяє визначати моделі як Python-класи, що автоматично відображаються на таблиці в базі даних. Він підтримує різноманітні типи полів, такі як CharField, IntegerField, DateTimeField, BooleanField, а також поля для зв'язків: ForeignKey, ManyToManyField, OneToOneField. QuerySets надають потужний API для виконання запитів до бази даних, усуваючи необхідність писати прямі SQL-запити. Система міграцій автоматично керує змінами схеми бази даних на основі модифікацій у моделях.

Представлення (View) обробляє HTTP-запити. Вони можуть бути реалізовані як функціональні представлення, що є простими Python-функціями, які приймають HttpRequest і повертають HttpResponse, або як класові представлення. Класові представлення надають структурований та перевикористовуваний підхід до написання логіки, а Django пропонує велику кількість вбудованих класових представлень, таких як ListView, DetailView, CreateView, UpdateView, DeleteView для типових операцій. Маршрутизатор URL забезпечує елегантне зіставлення URL-адрес із відповідними представленнями.

Шаблон (Template), що базується на Django Templates, відповідає за відображення даних користувачеві. Це HTML-розмітка з вбудованою логікою рендерингу.

Додатковими перевагами Django є автоматично генерована адміністративна панель, яка надає зручний веб-інтерфейс для операцій створення, читання, оновлення та видалення даних і має широкі можливості для кастомізації. Система форм спрощує створення HTML-форм, валідацію введених користувачем даних та їх обробку. Middleware дозволяє обробляти запити та відповіді на глобальному рівні, розширюючи функціонал застосунку. Сигнали забезпечують механізм для відправників сповіщати отримувачів про певні системні події, наприклад, збереження моделі або логін користувача. Вбудований Caching Framework підтримує різні бекенди кешування,

включаючи файлову систему, базу даних, Memcached та Redis, для оптимізації продуктивності. Інструменти Internationalization and Localization дозволяють створювати багатомовні та мультирегіональні веб-застосунки. Хоча основний інтерфейс буде реалізовано за допомогою Django Templates, важливо відзначити Django REST framework (DRF) – надзвичайно потужну та гнучку бібліотеку для створення REST API поверх Django. DRF спрощує серіалізацію даних, створення кінцевих точок, автентифікацію та авторизацію для API, що є цінним для майбутнього розширення системи, наприклад, для інтеграції з мобільними додатками або сторонніми сервісами.

2.3.4 Архітектурні підходи та API в контексті Django

Django-застосунки часто починаються як моноліти, де весь код знаходиться в одному проекті. Вбудована концепція "Django apps" дозволяє логічно структурувати проект на модулі, кожен з яких відповідає за певну частину функціоналу. Ця модульність полегшує розробку та підтримку. З часом, при зростанні навантаження та складності, такі логічні модулі можуть бути винесені в окремі мікросервіси, якщо це буде виправдано. Django REST framework надає всі необхідні інструменти для створення якісних REST API, що є основою для мікросервісної архітектури або взаємодії з клієнтськими SPA.

2.4 Системи управління базами даних (СУБД)

Вибір системи управління базами даних є критично важливим і залежить від типу даних, вимог до продуктивності, надійності, масштабованості та консистентності застосунку.

2.4.1 Реляційні СУБД (SQL)

Реляційні бази даних зберігають дані в структурованих таблицях та забезпечують ACID-транзакції (Atomicity, Consistency, Isolation, Durability). PostgreSQL обрано для даного проєкту. Його ключові особливості включають повну відповідність стандартам SQL, підтримку ACID-транзакцій та MVCC (Multiversion Concurrency Control) для ефективної обробки паралельних запитів, що забезпечує високу надійність та стабільність. PostgreSQL відзначається своєю розширюваністю, дозволяючи створювати власні типи даних, функції та оператори. Він підтримує такі мови як PL/pgSQL та PL/Python, пропонує потужні можливості індексації, повнотекстовий пошук та підтримку геопросторових даних через розширення PostGIS. Окрім стандартних типів даних (INTEGER, VARCHAR, TIMESTAMP), PostgreSQL підтримує масиви, діапазони, hstore та JSON/JSONB. Тип JSONB особливо корисний, оскільки дозволяє ефективно зберігати та індексувати JSON-документи в бінарному форматі.

Вибір PostgreSQL для універсальної системи бронювання обґрунтований кількома факторами. Надійність та ACID-транзакції є критично важливими для фінансових операцій, пов'язаних з бронюванням та оплатою квитків, гарантуючи цілісність даних. Використання JSONB надає необхідну гнучкість, дозволяючи системі підтримувати різні типи подій зі специфічними атрибутами без постійної зміни схеми основних таблиць. Геопросторові можливості відкривають потенціал для реалізації функціоналу пошуку подій за географічним розташуванням або в певному радіусі в майбутньому. Крім того, відмінна інтеграція з Django ORM спрощує розробку та взаємодію з базою даних.

Серед інших популярних реляційних СУБД варто згадати MySQL/MariaDB, відомі своєю швидкістю на операціях читання та широким розповсюдженням, а також SQLite — легковисну файлову СУБД, ідеальну для

локальної розробки та тестування завдяки простоті налаштування та переносимості. Django використовує SQLite за замовчуванням для розробки.

2.4.2 Нереляційні СУБД (NoSQL)

NoSQL бази даних пропонують альтернативні моделі даних та часто кращу горизонтальну масштабованість для певних сценаріїв використання.

MongoDB зберігає дані у вигляді BSON-документів. Її гнучка схема дозволяє легко змінювати структуру даних, що корисно для проєктів зі швидкозмінними або непередбачуваними структурами даних. MongoDB добре масштабується горизонтально за допомогою шардингу.

Redis є in-memory сховищем типу "ключ-значення", що забезпечує надзвичайно високу швидкість читання та запису. Він підтримує різні структури даних, такі як рядки, списки, множини, впорядковані множини, хеші та потоки. Redis також може зберігати дані на диск для забезпечення персистентності.

2.5 Технології інфраструктури та DevOps

Розгортання, управління та підтримка веб-застосунку вимагають надійної інфраструктури та ефективних DevOps-практик.

2.5.1 Хмарні платформи

Провідні хмарні провайдери, такі як Amazon Web Services (AWS), Google Cloud Platform (GCP) та Microsoft Azure, пропонують широкий спектр сервісів, що охоплюють обчислювальні ресурси, керовані бази даних, сховища даних та платформи як сервіс. Вони надають доступ до віртуальних машин, наприклад,

EC2 (AWS), Compute Engine (GCP) або Azure VMs, для розгортання застосунків. Для управління базами даних доступні такі керовані сервіси, як RDS (AWS), Cloud SQL (GCP) та Azure SQL Database, що спрощують адміністрування. Зберігання даних забезпечується сервісами на кшталт S3 (AWS), Cloud Storage (GCP) та Azure Blob Storage. Крім того, ці платформи пропонують Platform as a Service (PaaS) рішення, зокрема AWS Elastic Beanstalk, Google App Engine та Azure App Service, які значно спрощують розгортання та управління Django-застосунками, автоматизуючи багато аспектів інфраструктури.

2.5.2 Контейнеризація

Docker: Стандарт для створення, розгортання та запуску застосунків у ізольованих середовищах – контейнерах. Dockerfile описує, як зібрати образ застосунку, включаючи всі залежності. Docker Compose використовується для управління багатоконтейнерними застосунками на локальній машині розробника (наприклад, контейнер з Django-застосунком, контейнер з PostgreSQL). Контейнеризація забезпечує консистентність середовища на всіх етапах.

Kubernetes (K8s): Платформа з відкритим кодом для автоматизації розгортання, масштабування та управління контейнеризованими застосунками. Для великих, високонавантажених систем Kubernetes є де-факто стандартом.

2.5.3 CI/CD та GitHub Workflows

CI/CD (Continuous Integration/Continuous Delivery): Комплекс практик для автоматизації інтеграції коду від різних розробників, його автоматичного тестування та розгортання застосунку. Такий підхід значно прискорює релізи, підвищує якість коду та мінімізує ризики в процесі розробки.

GitHub Workflows: Інтегрована система автоматизації, що дозволяє створювати власні робочі процеси безпосередньо в репозиторії проєкту. Workflows визначаються у YAML-файлах, розташованих в директорії .github/workflows. Вони запускаються за допомогою Events, які є тригерами, такими як push у гілку, створення pull_request, запуск за розкладом (schedule) або ручний запуск (workflow_dispatch). Кожен workflow складається з одного або кількох Jobs — наборів кроків, що виконуються на віртуальних машинах, званих ранерами. Jobs можуть виконуватися паралельно або послідовно, а Steps є окремими командами або діями в рамках job. Для розширення функціоналу використовуються Actions — перевикористовувані одиниці коду, доступні на GitHub Marketplace у великій кількості.

Вибір GitHub Workflows для цього проєкту зумовлений його тісною інтеграцією з GitHub, де зберігатиметься код. Це усуває потребу в налаштуванні сторонніх сервісів, роблячи процес CI/CD більш природним та зручним.

Для дипломного проєкту простота налаштування для базових завдань є ключовою. GitHub Workflows дозволяє легко автоматизувати запуск тестів, лінтерів та, за необхідності, розгортання на хостинг-платформу. Важливим аспектом є вартість, оскільки GitHub Workflows надає щедрі безкоштовні ліміти для публічних і приватних репозиторіїв, що є значною перевагою для студентських проєктів.

Доступність готових Actions є ще однією перевагою. Існує велика кількість готових actions для налаштування середовища Python, встановлення залежностей, запуску Django-тестів, збірки Docker-образів та розгортання на популярні хмарні платформи.

Хоча існують інші потужні інструменти CI/CD, такі як GitLab CI/CD, інтегрований у GitLab, та Jenkins, який дуже гнучкий, але вимагає більше налаштувань та власної інфраструктури, для цього проєкту GitHub Workflows пропонує оптимальний баланс простоти та функціональності.

2.5.4 Моніторинг та логування

Моніторинг: Збір та аналіз метрик про роботу системи: навантаження на CPU та пам'ять, час відповіді, кількість помилок. Інструменти: Prometheus, Grafana, Zabbix, Datadog.

Логування: Збір, зберігання та аналіз логів з усіх компонентів системи для діагностики проблем. Інструменти: ELK Stack (Elasticsearch, Logstash, Kibana), EFK Stack (Elasticsearch, Fluentd, Kibana). Django має вбудовану систему логування, яку можна налаштувати для запису в файли або відправки в централізовані системи.

Відстеження помилок: Автоматичний збір інформації про винятки та помилки, що виникають у застосунку. Sentry є популярним інструментом, що добре інтегрується з Django та надає детальні звіти про помилки.

ВИСНОВОК ДО РОЗДІЛУ 2

У цьому розділі, після детального аналізу сучасних технологій веброзробки, обґрунтовано вибір технологічного стеку для універсального вебзастосунку з бронювання квитків. Це рішення було прийнято з урахуванням цілей дипломного проєкту, його часових обмежень та вимог до продуктивності, масштабованості й надійності.

Ключовими компонентами обраного стеку стали Python 3.12 у поєднанні з фреймворком Django для бекенду. Таке рішення забезпечує швидкість розробки, гнучкість та надійну архітектуру. Для фронтенду обрано Django Templates, що дозволяє ефективно створювати інтерфейс із необхідною інтерактивністю за допомогою JavaScript. PostgreSQL використовуватиметься як основна реляційна база даних, гарантуючи цілісність транзакцій та гнучкість у роботі з даними. Управління розробкою та автоматизація процесів CI/CD покладені на Git та GitHub Workflows.

Обраний стек є збалансованим і прагматичним рішенням. Він поєднує перевірені часом, потужні інструменти, які забезпечують швидкий старт проєкту, створюють міцну основу для основного функціоналу та надають можливості для подальшого масштабування. Автоматизація через GitHub Workflows додатково оптимізує процес розробки, підвищуючи якість коду та ефективність. Цей технологічний вибір формує надійну та адекватну основу для успішної реалізації дипломного проєкту.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Опис процесу розробки та застосована методологія

Цей розділ присвячений безпосередньому опису процесу створення універсального веб-застосунку для бронювання квитків на події. Він деталізує етапи розробки програмного продукту, включаючи проєктування бази даних, реалізацію бекенд та фронтенд логіки з використанням обраних технологій, інтеграцію платіжної системи, а також налаштування середовища розробки та інструментів автоматизації.

Для розробки проєкту обрано гнучку ітеративну методологію, яка дозволяє послідовно реалізовувати функціонал невеликими, керованими частинами. Цей підхід забезпечує поступове нарощування можливостей системи, регулярний аналіз результатів та внесення необхідних коректив. Основні етапи розробки охоплювали детальне проєктування, що включало уточнення вимог, архітектуру бази даних на основі моделей Django та визначення взаємодії модулів. Налаштування середовища розробки проводилося за допомогою Docker та Docker Compose для створення ізольованого та відтворюваного середовища. Реалізація функціоналу включала створення моделей даних, механізмів аутентифікації, відображення списку та деталей подій, а також основного бізнес-процесу бронювання квитків, управління замовленнями та інтеграції з платіжною системою Stripe. Розробка інтерфейсу користувача відбувалася шляхом створення шаблонів Django, а налаштування CI/CD забезпечувало автоматизацію процесів інтеграції за допомогою GitHub Workflows. Завершальним етапом була ручна перевірка функціоналу системи. Цей підхід дозволив гнучко керувати процесом розробки та послідовно створювати функціональний програмний продукт.

3.2 Налаштування середовища та інструменти розробки

Для забезпечення ефективного та відтворюваного процесу розробки було ретельно налаштовано відповідне середовище та підібрано комплекс інструментів. Проєкт базується на Python 3.12 та використовує Django 5.2.1 як основний веб-фреймворк. В якості системи управління базами даних обрано PostgreSQL 15, що слугує як для розробки, так і для імітації "продакшн" середовища.

Розробка ведеться в Visual Studio Code (VS Code), доповненому необхідними розширеннями для Python та Django. Git застосовується для системи контролю версій, а GitHub слугує платформою для хостингу коду та інтеграції GitHub Workflows для CI/CD процесів.

Ключовим елементом налаштування стало використання контейнеризації. Docker задіяний для створення ізольованих контейнерів, що інкапсулюють застосунок та всі його залежності. Для цього розроблено Dockerfile, який визначає базовий образ Python, копіює файли проєкту, встановлює залежності з requirements.txt та налаштовує запуск WSGI-сервера. Docker Compose використовується для оркестрації багатоконтейнерних застосунків під час локальної розробки. Файл docker-compose.yml конфігурує сервіси, включаючи Django-застосунок та базу даних PostgreSQL, встановлюючи їх взаємодію, мережеві налаштування та томи для зберігання даних. Це дозволило спростити запуск і зупинку всього середовища розробки однією командою, забезпечуючи його консистентність та легкість перенесення. WSGI-сервером для розгортання в Docker є Gunicorn. Для обробки онлайн-платежів інтегровано платіжну систему Stripe. Впровадження Docker та Docker Compose суттєво покращило узгодженість середовища розробки та спростило процес адаптації для нових розробників або міграції проєкту.

3.3 Проектування бази даних

Структура бази даних є фундаментом для зберігання та управління інформацією в системі бронювання квитків. Вона була реалізована за допомогою Django ORM на основі Python-класів (моделей).

На етапі проектування була розроблена концептуальна модель даних, що візуалізує основні сутності та їхні взаємозв'язки. Ключові сутності включають Користувача, Місце Проведення, Подію та Квиток. Місце Проведення може містити багато Подій. Кожна Подія пов'язана з багатьма Квитками. Користувач може забронювати багато Квитків, і кожен Квиток (якщо він заброньований) належить одному Користувачеві.

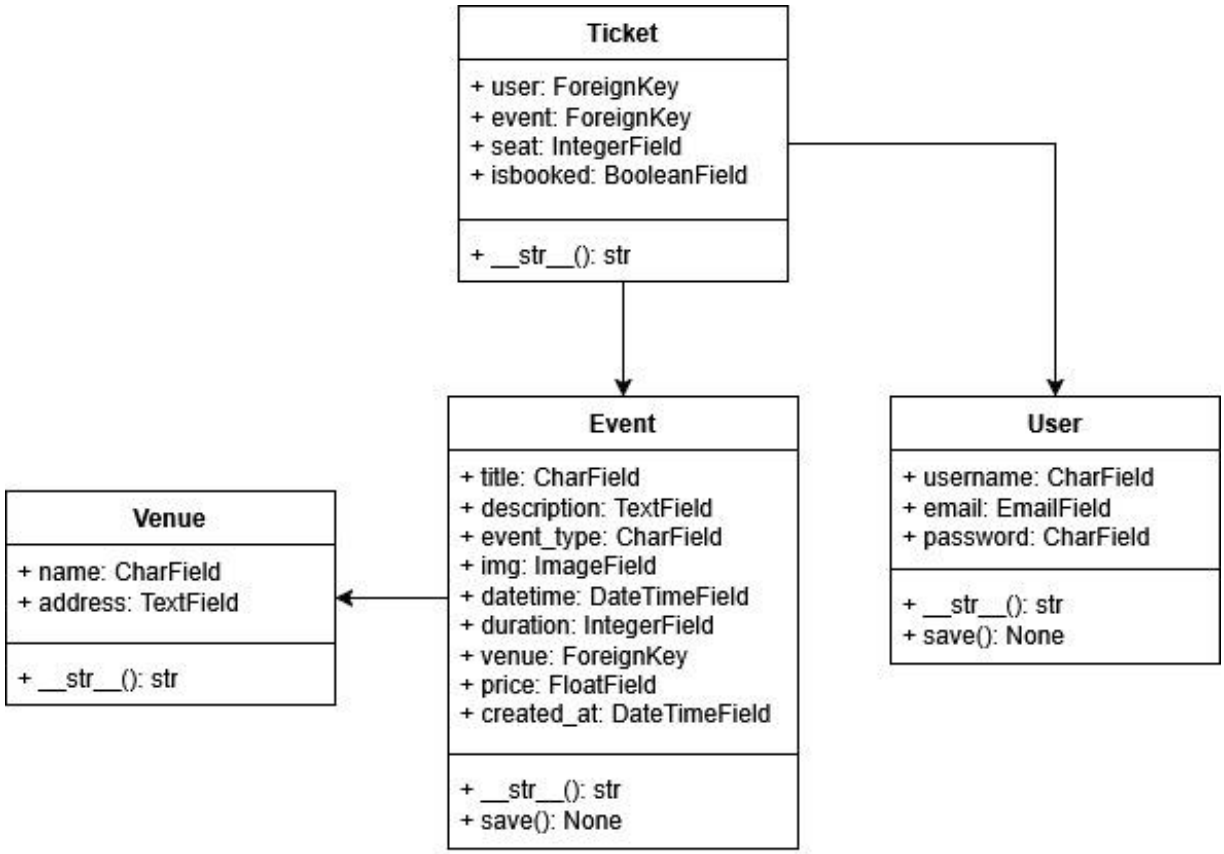


Рисунок 3.1 – Схема Django моделей

Модель Venue зберігає інформацію про локації подій, включаючи name (назва локації, до 100 символів) та address. Метод __str__ цієї моделі повертає назву локації.

Центральною моделлю системи є Event, яка описує конкретний захід і має визначені типи подій (кіно, театр, концерт, виставка). Її основні поля включають title (назва, до 100 символів), description (опис, може бути порожнім), event_type (тип з визначених варіантів), img (зображення, шлях до якого динамічно генерується функцією upload_to за датою створення), datetime (дата та час проведення), duration (тривалість у хвилинах, за замовчуванням 0), venue (зовнішній ключ до моделі Venue, з on_delete=models.CASCADE), price (ціна квитка) та created_at (дата створення запису, встановлюється автоматично). Метод __str__ повертає назву події та її тип. Перевизначений метод save цієї моделі автоматично створює 90 екземплярів моделі Ticket для нової події з номерами місць від 0 до 89, реалізуючи специфічну бізнес-логіку проєкту про стандартну кількість місць.

Модель Ticket представляє окремий квиток на подію. Вона містить поля user (зовнішній ключ до стандартної моделі Django User, може бути порожнім, якщо квиток не заброньований), event (зовнішній ключ до моделі Event, з on_delete=models.CASCADE), seat (номер місця) та is_booked (булеве поле, за замовчуванням False, що вказує на статус бронювання). Метод __str__ повертає опис події та номер місця квитка.

Зв'язки між моделями чітко визначені за допомогою ForeignKey, забезпечуючи реляційну цілісність даних. Функція upload_to для поля img в моделі Event гарантує організоване зберігання зображень. Автоматичне створення екземплярів Ticket при збереженні Event є важливою особливістю бізнес-логіки, реалізованої безпосередньо на рівні моделі.

3.4 Розробка Backend частини

Backend частина є ядром системи, що реалізує всю бізнес-логіку, використовуючи потужності фреймворку Django.

3.4.1 Архітектурний патерн Model-View-Template (MVT) в Django

Django дотримується архітектурного патерну MVT (Model-View-Template), що є варіацією відомого патерну MVC (Model-View-Controller). Розуміння MVT є ключовим для ефективної розробки на Django, оскільки він забезпечує чітке розділення відповідальності між управлінням даними, логікою обробки запитів та їх представленням, роблячи код організованим, гнучким та легким для підтримки.

Model (Модель) є єдиним джерелом правдивої інформації про дані. Вона містить поля та логіку для взаємодії з даними, що зберігаються в базі даних. Django ORM забезпечує цю взаємодію, абстрагуючи SQL-запити. У цьому проєкті, Venue, Event, Ticket є прикладами моделей.

View (Представлення) виконує роль обробника запитів, подібну до "Controller" у патерні MVC. Він приймає HTTP-запит, виконує необхідну бізнес-логіку — наприклад, отримує або зберігає дані через моделі, обробляє дані форм, виконує обчислення — а потім повертає HTTP-відповідь. Це може бути HTML-сторінка, згенерована за допомогою шаблону, перенаправлення або документ JSON/XML.

Template (Шаблон) відповідає за представлення даних користувачеві. Це текстовий файл, зазвичай HTML, що містить спеціальну розмітку Django Template Language (DTL). Шаблони визначають структуру вихідного документа, а динамічні дані підставляються в них з контексту, переданого представленням.

Концептуально, процес за патерном MVT відбувається так: користувач надсилає HTTP-запит до певної URL-адреси. URL Dispatcher Django аналізує URL і спрямовує запит до відповідного View. View отримує запит, взаємодіє з Model для отримання чи збереження даних або виконання бізнес-логіки. Після обробки логіки, View обирає відповідний Template, передає йому дані у вигляді контексту. Template Engine Django обробляє шаблон, підставляючи дані та генеруючи фінальну HTML-сторінку, яка потім повертається як HTTP-відповідь користувачеві.

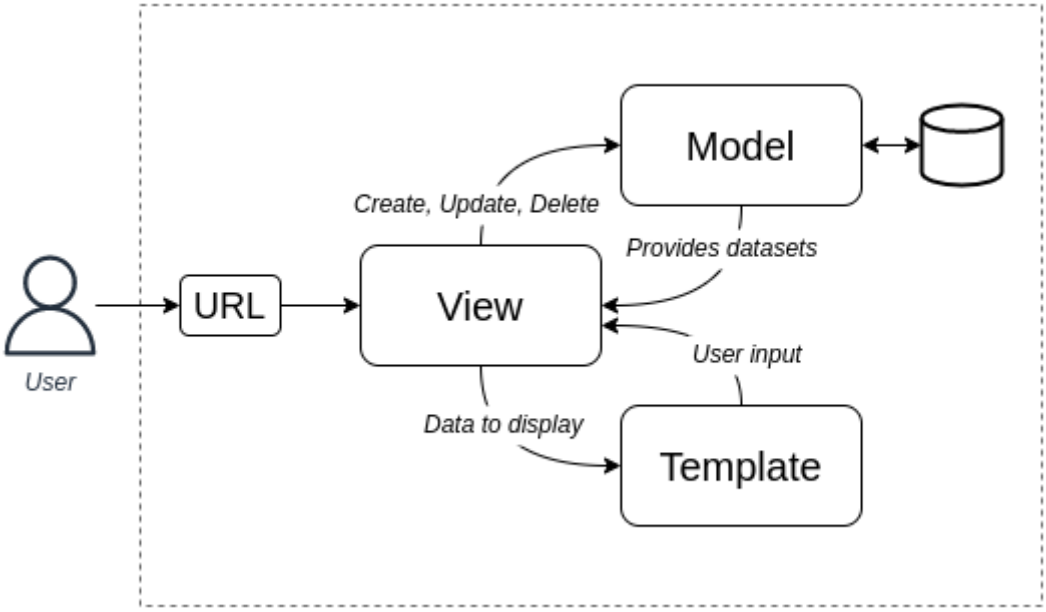


Рисунок 3.2 – структурна схема MVT патерну в Django

3.4.2 Структура Django-проекту та додатків (apps)

Проект Django був організований з використанням додатків (apps) для логічного групування функціоналу:

- ticket_project/: Основний конфігураційний додаток (містить settings.py, urls.py проекту).
- main/: Додаток для управління сутностями Venue, Event, Ticket.
- payments/: Для обробки логіки бронювання та управління замовленнями.

3.4.3 Реалізація представлень (Views) та бізнес-логіки

Файл views.py містить представлення, що обробляють запити та реалізують основну бізнес-логіку застосунку.

Розроблено представлення для відображення списку всіх опублікованих подій, що включає функціонал пошуку за назвою. Також реалізовано представлення детальної сторінки події, яка надає повну інформацію про конкретний захід: опис, зображення, деталі місця проведення, а також список доступних для бронювання квитків чи місць.

Для реєстрації та аутентифікації користувачів використовуються стандартні засоби django.contrib.auth, що забезпечують безпечний вхід, вихід та реєстрацію.

Процес бронювання квитка включає кілька етапів: користувач обирає подію та конкретне місце. Система здійснює перевірку статусу квитка, а саме, чи не заброньований він вже (за допомогою Ticket.is_booked). Якщо квиток вільний, його статус змінюється на is_booked=True, і він прив'язується до поточного користувача (ticket.user = request.user). Після цього користувач перенаправляється на сторінку оплати через платіжну систему Stripe.

3.4.4 Налаштування маршрутизації

Маршрутизація URL-адрес до відповідних представлень налаштовувалася у файлах urls.py кожного додатку та в основному urls.py проекту, використовуючи функцію path().

3.4.5 Конфігурація адміністративної панелі Django Admin

Стандартна адміністративна панель Django була налаштована для зручного управління даними: Venue, Event, Ticket. Це дозволяє адміністраторам

системи легко додавати, редагувати та видаляти події, місця проведення та переглядати інформацію про квитки та їх статус.

3.4.6 Інтеграція платіжної системи Stripe

Для забезпечення безпечної та ефективної обробки онлайн-платежів за заброньовані квитки до веб-застосунку інтегровано платіжну систему Stripe. Інтеграція реалізована через окремий Django-додаток payments, що забезпечує модульність та чітке розділення відповідальності. Цей підхід дозволяє використовувати Stripe Checkout — готове рішення для прийому платежів, яке значно спрощує розробку та гарантує високий рівень безпеки, оскільки чутливі платіжні дані ніколи не проходять через сервери застосунку.

Процес інтеграції включав кілька ключових етапів. Етап налаштування Stripe включає в себе створення та налаштування облікового запису розробника в системі Stripe. Отримання публічного та секретного API-ключів, які використовуються для аутентифікації запитів до API Stripe з клієнтської та серверної частини відповідно. А також налаштування секретного ключа вебхука (Webhook Secret) для перевірки достовірності подій, що надходять від Stripe.

На клієнтській частині користувач взаємодіє з інтерфейсом бронювання. Після підтвердження замовлення надсилається запит на створення платіжної сесії до сервера. У відповідь клієнтська частина отримує URL-адресу для перенаправлення на сторінку Stripe Checkout. Весь процес введення платіжних даних відбувається безпосередньо на захищеній сторінці Stripe Checkout, що мінімізує ризики безпеки для самого застосунку.

На серверній частині, у межах Django-додатка payments, реалізовані дві основні функції: create_checkout_session та stripe_webhook. Функція create_checkout_session приймає POST-запит від клієнтської частини з ідентифікаторами обраних квитків. Після перевірки наявності та доступності

квитків, формується список товарних позицій для сесії Stripe Checkout, де ціна передається в копійках, відповідно до вимог Stripe. Через Stripe API створюється об'єкт checkout.Session із зазначенням URL-адрес для успішного завершення та скасування платежу, типу платежу та метаданих, що включають ідентифікатори квитків та ID користувача. Ці метадані критично важливі для подальшої обробки результатів платежу. У відповідь клієнту надсилається URL-адреса створеної сесії Stripe Checkout, на яку користувач буде перенаправлений.

Функція stripe_webhook є кінцевою точкою для отримання асинхронних подій від Stripe. Після успішного завершення платежу на стороні Stripe, система надсилає подію checkout.session.completed на цей вебхук. Отриманий payload та підпис запиту перевіряються за допомогою stripe.Webhook.construct_event для забезпечення безпеки та автентичності події. У разі успішної події checkout.session.completed з об'єкта сесії витягуються збережені метадані – ідентифікатори квитків та ID користувача. Для кожного ідентифікатора квитка в базі даних оновлюється статус квитка на "заброньовано" (is_booked = True) та присвоюється ідентифікатор користувача, який здійснив покупку. У разі помилки, повертається відповідний статус. Зберігання ідентифікатора транзакції Stripe, наприклад, PaymentIntent ID, у базі даних забезпечує можливість відстеження платежів та спрощує обробку можливих повернень коштів, при цьому повні дані платіжних карток не зберігаються, що відповідає стандартам безпеки PCI DSS.

Цей підхід з використанням Stripe Checkout та вебхуків забезпечує високий рівень безпеки платіжних операцій, мінімізуючи обсяг чутливих даних, що проходять через сервери застосунку, та спрощує інтеграцію платіжного функціоналу.

3.5 Розробка Frontend частини

Frontend частина реалізована з використанням вбудованого шаблонного рушія Django.

3.5.1 Структура шаблонів та успадкування

Була створена ієрархічна структура шаблонів. Основний шаблон base.html визначав загальний макет сторінки (шапка, навігація, підвал, підключення CSS/JS). Специфічні сторінки (список подій, деталі події, форма входу) успадковували цей базовий шаблон, перевизначаючи його блоки ({% block content %}, {% block title %}). Використовувалися часткові шаблони ({% include %}) для перевикористовуваних елементів.

3.5.2 Відображення динамічних даних у шаблонах

Дані, передані з представлень у контексті, відображалися в шаблонах за допомогою змінних (напр., {{ event.title }}, {{ ticket.seat }}) та тегів (напр., {% for event in events_list %}). Фільтри Django Templates використовувалися для форматування даних (напр., дати, тексту).

3.5.3 Робота зі статичними файлами

Статичні файли (CSS для стилізації, JavaScript для клієнтської логіки, зображення) управлялися за допомогою стандартних засобів Django. Вони розміщувалися у відповідних директоріях static/ та підключалися в шаблонах через тег {% static %}.

3.5.4 Приклади використання JavaScript для покращення інтерактивності

Клієнтський JavaScript був інтегрований для підвищення інтерактивності користувацького інтерфейсу, доповнюючи серверний рендеринг Django Templates. Його застосування дозволило реалізувати динамічну взаємодію з елементами сторінки, надаючи миттєвий зворотний зв'язок користувачу. Також JavaScript використовувався для асинхронних AJAX-запитів, що дозволило оновлювати частини сторінки, наприклад, перевіряти статус місця, без необхідності повного перезавантаження.

3.6 Реалізація ключових алгоритмів та логіки

Генерація квитків для події: Як описано в моделі Event, при збереженні нової події автоматично генерується 90 квитків (місць) для неї, якщо вони ще не були створені. Це спрощує управління квитками для подій зі стандартною кількістю місць.

Логіка бронювання місця: При спробі забронювати квиток (Ticket), система перевіряє поле isbooked. Якщо воно False, квиток позначається як isbooked=True і пов'язується з request.user. Ця операція має бути атомарною в високонавантаженій системі, щоб уникнути подвійного бронювання одного й того ж місця. У Django це можна забезпечити за допомогою select_for_update() в транзакції.

Обробка різних типів подій: Хоча модель Event має поле event_type, логіка відображення та бронювання для різних типів (кіно, театр, концерт, виставка) могла б відрізнятися на рівні шаблонів та, можливо, представлень, особливо якщо б для театральних або концертних подій реалізовувалася

складна схема залу. У даній структурі моделей, основна відмінність типів подій є інформативною, а механізм квитків/місць уніфікований (90 місць).

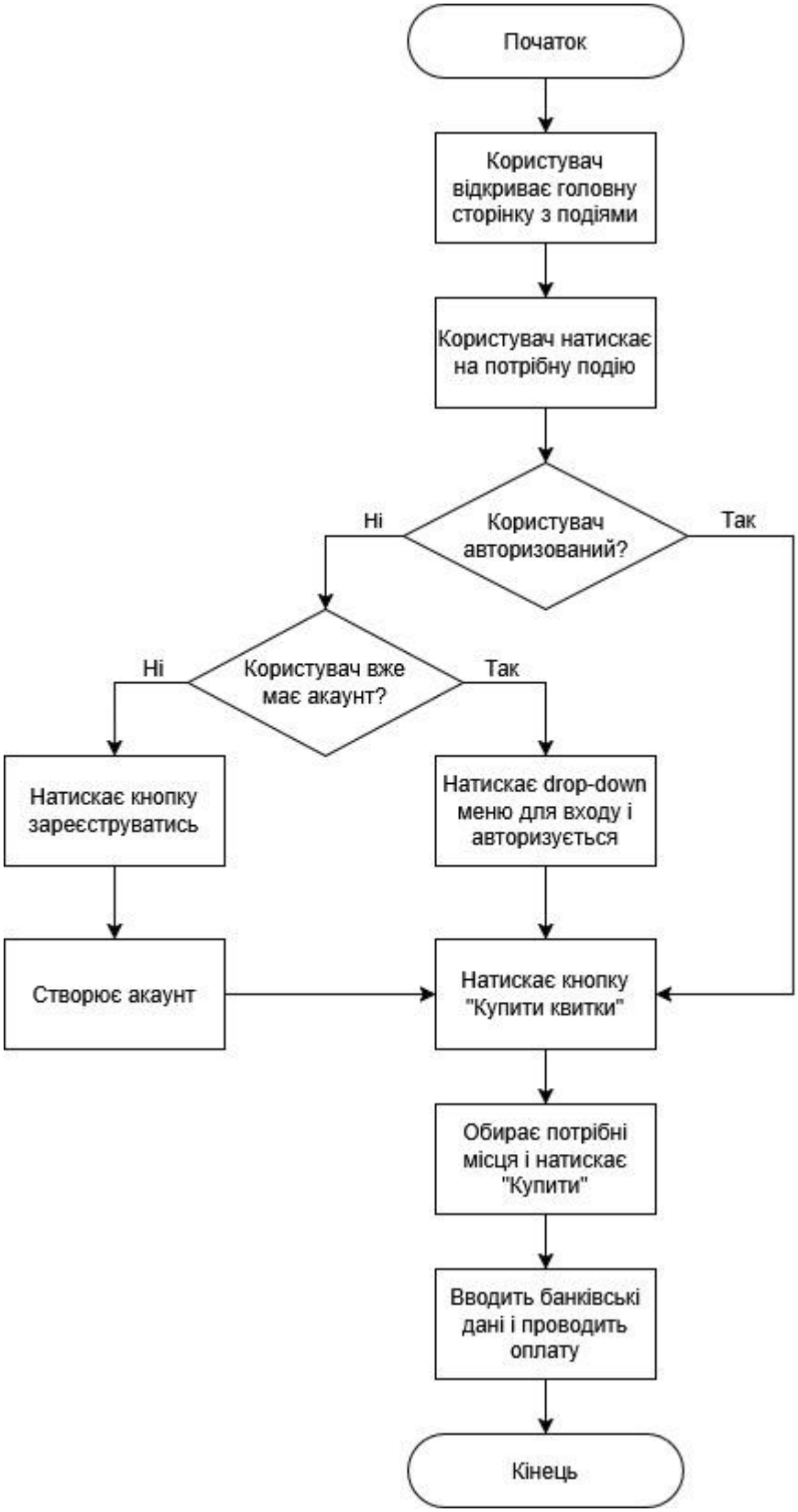


Рисунок 3.3 – Алгоритм дій веб-застосунку

3.7 Аспекти безпеки застосунку

Безпека є пріоритетним аспектом розробки застосунку. Для її забезпечення використовувалися стандартні, перевірені засоби захисту, що надаються Django. Серед них — вбудований захист від поширених веб-вразливостей, таких як CSRF (Cross-Site Request Forgery), XSS (Cross-Site Scripting) та SQL Injection, який автоматично забезпечується Django ORM та шаблонним рушієм.

Для захисту даних користувачів реалізовано безпечне хешування паролів. Системи аутентифікації та авторизації побудовані на базі django.contrib.auth, що дозволяє обмежувати доступ до певних функцій для неавторизованих користувачів, наприклад, за допомогою декоратора @login_required.

Особлива увага приділена безпечній обробці платіжних даних через інтеграцію зі Stripe. Це рішення мінімізує ризики для застосунку, оскільки чутливі фінансові дані обробляються безпосередньо на захищених серверах платіжної системи.

3.8 Налаштування CI/CD за допомогою GitHub Workflows

Для автоматизації процесів інтеграції коду було налаштовано GitHub Workflows. Основний workflow-файл, що описує робочий процес, створено в директорії проєкту .github/workflows/.

Ключові кроки цього workflow включають: тригер, що запускає процес при кожному push в основну гілку та при створенні або оновленні pull_request; checkout, який відповідає за отримання коду проєкту; налаштування версії Python; встановлення залежностей проєкту за допомогою Poetry; та запуск лінтерів, зокрема статичного аналізатора коду Flake8, для перевірки стилю та потенційних помилок.

Головна перевага цього підходу полягає в автоматичному запуску перевірок при кожній зміні коду, що допомагає підтримувати його високу якість та консистентність протягом усього циклу розробки.

ВИСНОВОК ДО РОЗДІЛУ 3

У цьому розділі, зосередженому на практичній реалізації, було успішно описано та втілено процес розробки універсального вебзастосунку для бронювання квитків. Завдяки використанню Docker та Docker Compose забезпечено консистентне середовище розробки, а проєктування бази даних через Django моделі заклало надійний фундамент для зберігання інформації. Застосований архітектурний патерн MVT дозволив чітко розділити відповідальність та сприяти модульності коду. Основний функціонал застосунку, зокрема перегляд подій, реєстрація/аутентифікація та повний цикл бронювання квитків з інтеграцією Stripe для безпечних платежів, був реалізований. Налаштована адміністративна панель Django значно спрощує управління контентом.

Хоча автоматизоване тестування не було основним завданням, ручна перевірка ключових сценаріїв у поєднанні з базовою автоматизацією через GitHub Workflows допомогла підтримувати якість коду.

В результаті, розроблений програмний продукт повністю відповідає поставленим завданням, демонструючи ефективне застосування обраних технологій – Python, Django, PostgreSQL та Docker. Створена архітектура є гнучкою, що забезпечує міцну основу для майбутнього розвитку та розширення функціоналу системи.

РОЗДІЛ 4. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОГО ВЕБ-ЗАСТОСУНКУ

4.1 Цілі та напрямки аналізу результатів розробки

Даний розділ присвячений всебічному дослідженню та аналізу результатів практичної розробки універсального веб-застосунку для бронювання квитків, процес створення якого було описано у Розділі 3. Основна мета цього аналізу – об'єктивно оцінити реалізований програмний продукт, його функціональні можливості, зручність використання та відповідність першочерговим цілям, поставленим на етапі проектування. Буде проведено оцінку ефективності реалізації ключових користувацьких сценаріїв, проаналізовано досвід взаємодії з користувацьким інтерфейсом, а також розглянуто, наскільки вдало обраний технологічний стек (Python, Django, PostgreSQL, Docker, Stripe) впорався із поставленими завданнями в контексті даного проекту. Результати цього дослідження дозволять не лише підсумувати досягнення, але й виявити сильні та слабкі сторони розробленого рішення, а також сформулювати обґрунтовані пропозиції щодо його потенційного подальшого розвитку.

4.2 Загальний опис реалізованого веб-застосунку та його архітектури

Розроблений веб-застосунок є функціональною платформою, що надає користувачам можливість переглядати афішу подій, шукати конкретні заходи та здійснювати купівлю квитків онлайн. Система спроектована як універсальна, тобто призначена для підтримки різних типів подій, хоча поточна реалізація моделі квитків (автоматична генерація 90 місць) є специфічною та може

слугувати основою для більш гнучких рішень у майбутньому. Архітектурно застосунок побудований на базі фреймворку Django, дотримуючись патерну MVT (Model-View-Template), що забезпечило чітке розділення логіки даних, обробки запитів та представлення. Взаємодія з базою даних PostgreSQL здійснюється через Django ORM. Інтеграція з платіжною системою Stripe забезпечує обробку фінансових транзакцій. Середовище розробки та розгортання було контейнеризовано за допомогою Docker та Docker Compose. Користувацький інтерфейс реалізовано за допомогою Django Templates, що забезпечує серверний рендеринг сторінок. Основні доступні користувачеві розділи включають головну сторінку зі списком подій, сторінки деталей подій, функцію пошуку, систему реєстрації/авторизації та статичні інформаційні сторінки.

4.3 Аналіз реалізованого функціоналу

Дослідження функціональних можливостей застосунку проводилося шляхом виконання основних користувацьких сценаріїв.

4.3.1 Головна сторінка та відображення подій

Головна сторінка слугує вітриною застосунку, надаючи користувачам огляд актуальних подій. Події представлені у вигляді карток, кожна з яких містить ключову інформацію: назву, зображення та опис. Таке подання дозволяє користувачеві швидко ознайомитися з пропозиціями та обрати цікаві для себе заходи. Список подій динамічно формується на основі даних, що зберігаються в базі даних PostgreSQL. Важливою особливістю реалізації є використання вбудованого в Django класового представлення ListView для відображення подій, яке автоматично забезпечує пагінацію. Це означає, що при

великій кількості подій вони розподіляються на декілька сторінок, що запобігає перевантаженню однієї сторінки даними, покращує час її початкового завантаження та загальний досвід користувача. На сторінці присутні елементи навігації пагінації (наприклад, "наступна", "попередня", номери сторінок), що дозволяють користувачеві легко переглядати весь каталог подій.

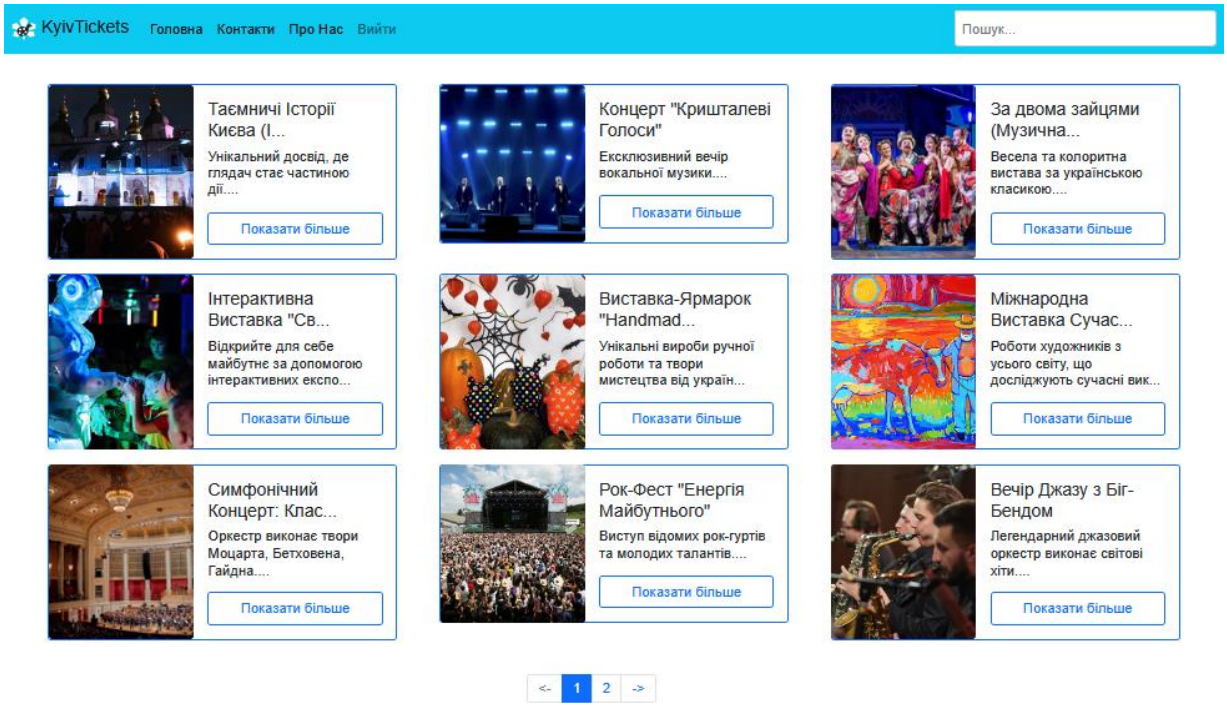


Рисунок 4.1 – Головна сторінка веб-застосунку

Окрім десктопної версії, дизайн головної сторінки адаптовано для мобільних пристроїв, забезпечуючи зручне та інтуїтивно зрозуміле відображення інформації.

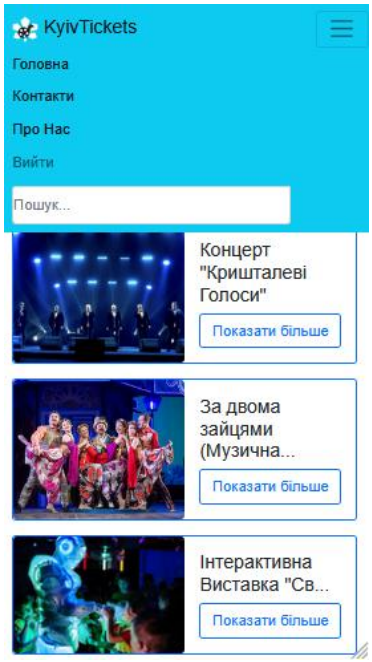


Рисунок 4.2 – Головна сторінка веб-застосунку (Мобільний варіант)

4.3.2 Функціонал пошуку подій

Для полегшення навігації та швидкого знаходження потрібних заходів у верхній частині сайту реалізовано пошукову строку. Користувач може ввести ключові слова (наприклад, назву події, частково назву) для здійснення пошуку. Пошук реалізовано на боці backend за допомогою Django ORM, який виконує запити до бази даних PostgreSQL по відповідному полю моделі Event. Результати пошуку відображаються у вигляді списку подій, що відповідають запиту. Якщо результатів пошуку багато, до них також може бути застосована пагінація, аналогічно до головної сторінки, для зручності перегляду.

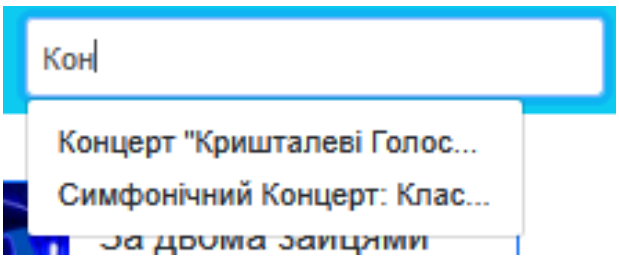


Рисунок 4.3 – Демонстрація роботи пошуку

4.3.3 Сторінка детальної інформації про подію

Перехід на сторінку деталізації події надає користувачеві повний набір необхідної інформації: розширений опис, всі характеристики заходу, інформацію про місце проведення та, що важливо, доступні квитки/місця. Відображення 90 автоматично згенерованих місць (якщо вони не забронювані) є зрозумілим, хоча і спрощеним підходом до візуалізації доступності. Структура сторінки логічна, інформація легко сприймається.

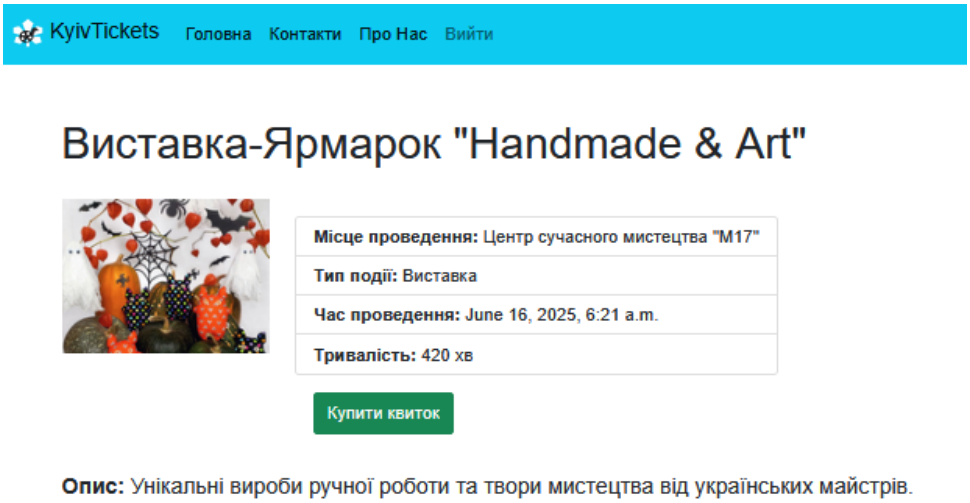


Рисунок 4.4 – Сторінка детальної інформації про подію

4.3.4 Процес купівлі квитків та інтеграція з Stripe

Це ядро функціоналу. Процес вибору квитка (місця) та переходу до оплати реалізовано послідовно.

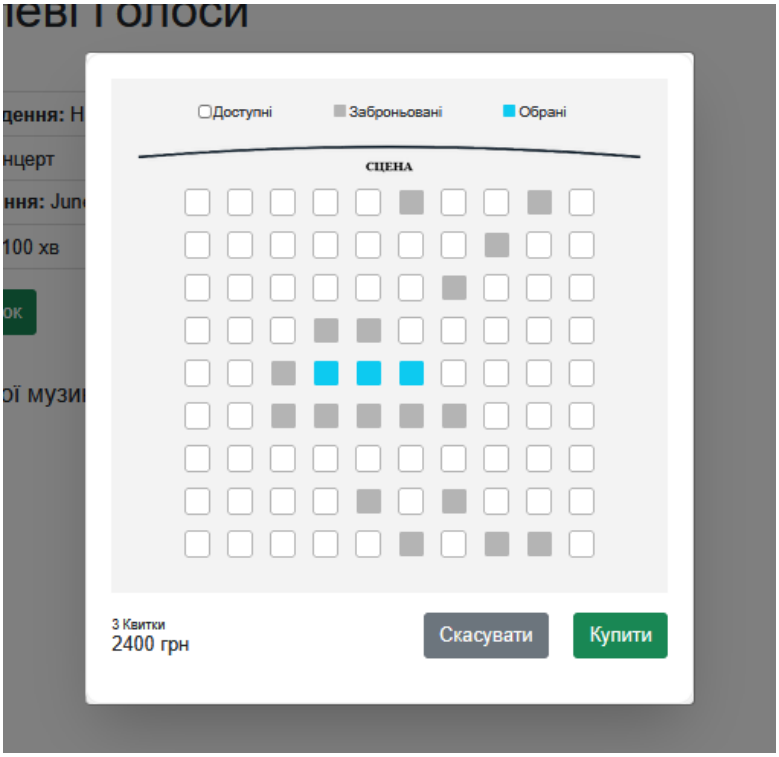


Рисунок 4.5 – Модальне вікно вибору місць для купівлі

Інтеграція з платіжною системою Stripe є ключовим аспектом розробленого функціоналу. Високий рівень безпеки платіжних операцій досягається завдяки використанню Stripe.js на клієнтській стороні: цей інструмент токенізує платіжні дані, унеможливлюючи потрапляння чутливої інформації про банківські картки на сервер застосунку, що повністю відповідає стандартам PCI DSS. Окрім надійного захисту, процес оплати через Stripe є стандартним та звичним для багатьох користувачів, що суттєво підвищує їхню довіру та забезпечує загальну зручність взаємодії. Надійність обробки транзакцій на сервері гарантується використанням офіційної бібліотеки Stripe; при цьому оновлення статусу квитка в базі даних (встановлення позначки is_booked=True та його прив'язка до конкретного користувача) відбувається

виключно після успішного підтвердження фінансової операції. Таким чином, реалізований механізм купівлі квитків через Stripe є не лише повністю функціональним, але й безпечним, що можна вважати важливим досягненням у рамках даного проекту.

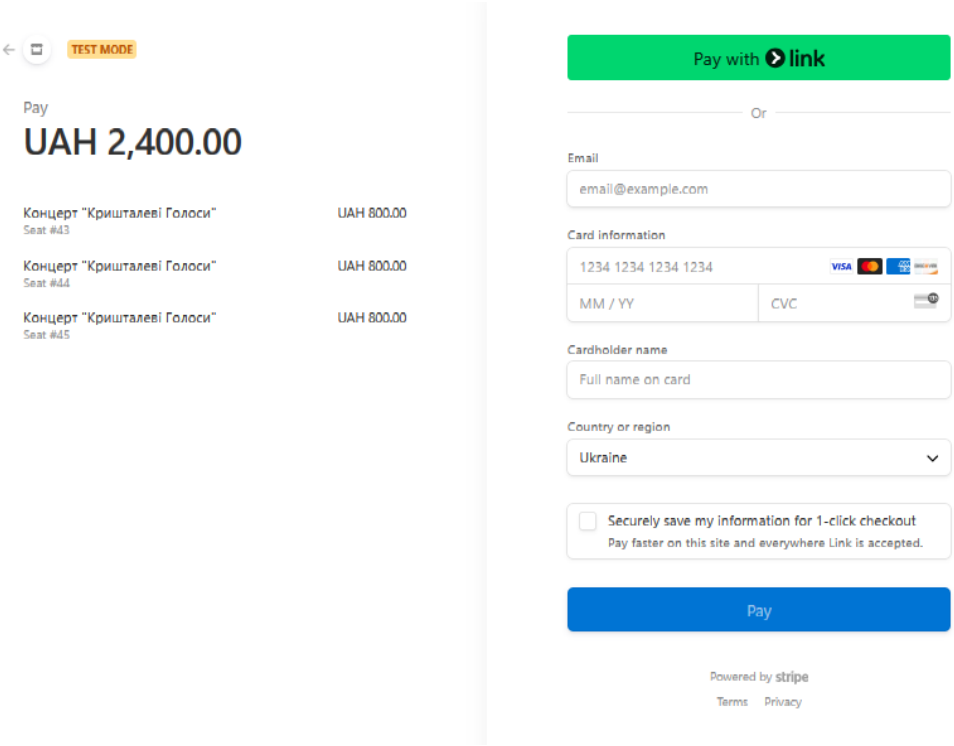


Рисунок 4.6 – Сторінка для проведення онлайн-оплати за допомогою платіжної системи Stripe

4.3.5 Система автентифікації та авторизації користувачів

Використання стандартного модуля django.contrib.auth для реєстрації та авторизації забезпечило надійний та перевірений часом функціонал. Форми реєстрації та входу є стандартними. Після авторизації користувач може здійснювати купівлю квитків, які потім асоціюються з його обліковим записом. Це закладає основу для майбутнього розширення функціоналу особистого кабінету (наприклад, перегляд історії бронювань). На даному етапі система коректно розрізняє авторизованих та неавторизованих користувачів, надаючи відповідні можливості.

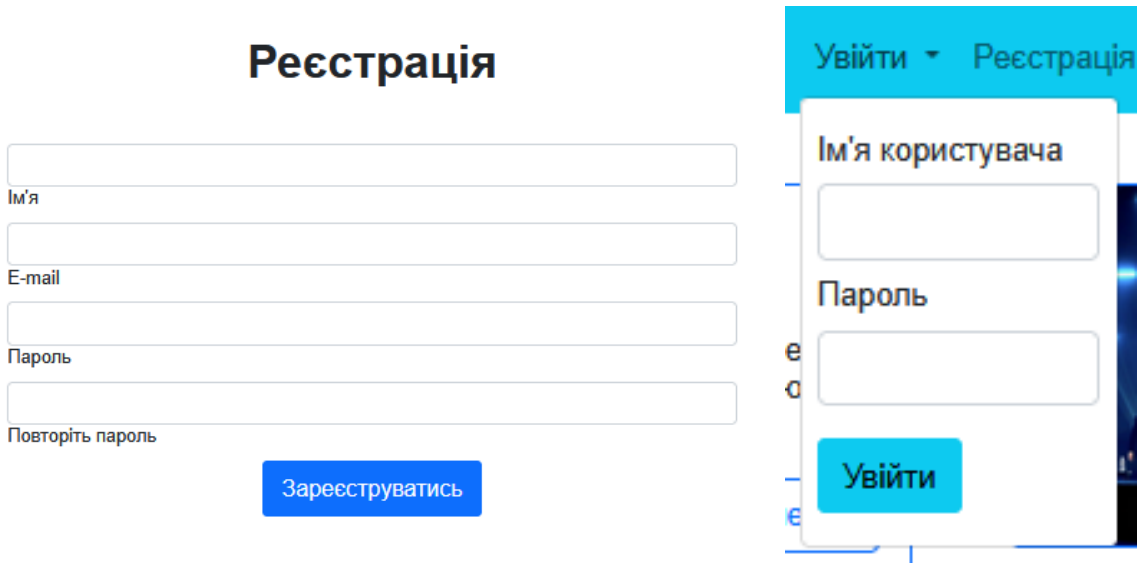


Рисунок 4.7 – Форми для реєстрації та авторизації

4.3.6 Інформаційні сторінки "Про нас" та "Контакти"

Ці сторінки успішно реалізовані та виконують свою інформаційну функцію, надаючи користувачам загальні відомості про проект та контактні дані. Їх реалізація за допомогою Django Templates є простою та ефективною.

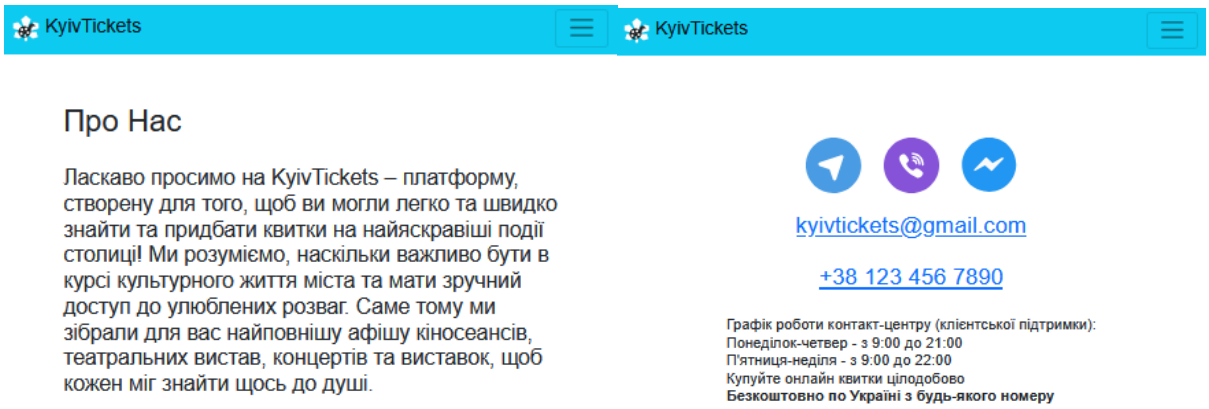


Рисунок 4.8 – Інформаційні сторінки "Про нас" та "Контакти"

4.4 Аналіз користувацького інтерфейсу (UI) та досвіду користувача (UX)

Оцінка користувацького інтерфейсу та досвіду взаємодії (UI/UX) базується на аналізі зручності роботи з реалізованим інтерфейсом. Загальний дизайн та навігація застосунку, побудовані на Django Templates, характеризуються простотою та функціональністю. Навігаційні елементи, такі як меню та посилання, розташовані логічно, що дозволяє користувачам легко переходити між основними розділами: головна сторінка, деталі події, сторінки контактів та "про нас", а також форми входу/реєстрації. Перевагою такого підходу є швидкість завантаження сторінок, за умови відсутності надмірно об'ємних ресурсів, та зрозуміла структура для користувача.

Зручність виконання ключових завдань оцінюється наступним чином: пошук та вибір події реалізовано ефективно завдяки пошуковій стрічці та візуальному представленню подій у вигляді карток, що спрощує знаходження та вибір заходів. Процес купівлі квитка, що охоплює вибір події, місця та оплату через Stripe, є послідовним та зрозумілим. Відсутність складної візуальної карти залу спрощує цей крок для поточного рівня реалізації, хоча й обмежує можливості для подій з традиційною схемою місць.

Однак, існують обмеження серверного рендерингу. Використання Django Templates означає, що більшість взаємодій призводять до повного або часткового перезавантаження сторінки. Це може сприйматися як менш плавний досвід порівняно з сучасними SPA-застосунками, особливо при частих фільтраціях або оновленнях даних. Втім, для основного сценарію "знайшов-купив" це не є критичним недоліком і вважається виправданим компромісом для дипломного проєкту. Загалом, користувацький досвід є задовільним для виконання основних функцій, а інтерфейс не перевантажений, що сприяє легкості його освоєння.

4.5 Оцінка використаного стеку технологій у практичній реалізації

Ефективність обраного технологічного стеку оцінювалася протягом усього процесу розробки та на основі фінального продукту.

Python та Django повністю підтвердили свою репутацію інструментів для швидкої розробки. Вбудовані компоненти Django, включаючи ORM, адмін-панель, систему автентифікації та шаблонізатор, значно прискорили реалізацію базового й основного функціоналу. Патерн MVT забезпечив відмінну структуру проєкту, а Python виявився зручною та ефективною мовою програмування.

PostgreSQL, як основна база даних, працювала стабільно, а гнучкість Django ORM дозволила ефективно взаємодіяти з нею без написання складних SQL-запитів для основного функціоналу.

Інтеграція Stripe пройшла успішно завдяки якісній документації та клієнтським бібліотекам. Це дозволило реалізувати надійний платіжний функціонал, уникнувши необхідності розробки власної складної та ризикованої системи обробки платежів.

Використання Docker та Docker Compose суттєво полегшило налаштування та управління середовищем розробки. Забезпечення однакового оточення для бази даних та веб-застосунку усунуло багато потенційних проблем, пов'язаних із конфігурацією та залежностями, що є особливо цінним при командній роботі. Це також закладає міцну основу для спрощеного розгортання на продакшн-сервери.

Загалом, обраний стек виявився добре збалансованим для завдань проєкту, забезпечивши необхідний рівень функціональності, надійності та швидкості розробки.

4.6 Відповідність поставленим завданням

Порівнюючи реалізований веб-застосунок з початковими цілями дипломного проєкту, можна констатувати, що основна мета — створення функціонального веб-застосунку для бронювання квитків — досягнута. Користувачі можуть переглядати події, здійснювати пошук і купувати квитки, а ключовий користувацький сценарій — від пошуку події до успішної оплати квитка через Stripe — повністю реалізовано.

Система автентифікації та базового профілювання користувачів функціонує належним чином, а застосунок містить необхідні інформаційні сторінки, що є стандартною практикою. Щодо універсальності, модель Event з полем event_type та гнучкими полями, такими як description та img, дозволяє додавати різні типи подій. Однак, специфічна логіка для кожного типу, наприклад, різні схеми місць або типи квитків, потребуватиме подальшого розвитку. Поточна реалізація з 90 автоматично генерованими "місцями" є уніфікованим, але спрощеним підходом.

Застосування сучасних технологій, таких як Python, Django, PostgreSQL, Docker, відповідає актуальним практикам веб-розробки. Загалом, можна стверджувати, що основні завдання, поставлені перед дипломним проєктом, були успішно виконані, і розроблено робочий прототип системи.

4.7 Обмеження та напрямки подальшого розвитку

Попри досягнуті результати, важливо об'єктивно оцінити обмеження поточної версії застосунку та визначити вектори для його подальшого вдосконалення.

Найбільш очевидним напрямком є розробка розширеної системи управління місцями та типами квитків. Це передбачає створення гнучкої

системи для подій з фіксованою розсадкою, включно з можливістю завантаження схем залів, визначенням різних цінових зон та інтерактивним вибором місць користувачем. Також необхідна функціональність для організаторів щодо визначення різних типів квитків, не лише за місцем, а й за категорією (наприклад, дитячий, студентський) з різними цінами та квотами.

Подальше вдосконалення включає деталізацію профілю користувача, додаючи історію бронювань, управління особистими даними та налаштування сповіщень. Критично важливим є розробка функціоналу для організаторів подій, що надасть їм окремий інтерфейс для самостійного додавання, редагування та управління своїми подіями, а також відстеження продажів квитків.

Система також потребує розширеного пошуку та фільтрації за датою, типом події, місцем проведення та ціною, з потенційною інтеграцією Elasticsearch для потужнішого повнотекстового пошуку. Додавання системи відгуків та рейтингів дозволить користувачам ділитися враженнями, а сповіщення (email або push) забезпечать своєчасну комунікацію щодо купівлі, нагадувань про події чи зміни у заходах.

У напрямку UI/UX розглядається можливість використання більш динамічних фронтенд-технологій, таких як поступова інтеграція React/Vue для покращення інтерактивності, та адаптація дизайну для кращого вигляду на різних пристроях. Для розширення аудиторії передбачається локалізація та інтернаціоналізація, включаючи підтримку кількох мов інтерфейсу та, можливо, валют. Нарешті, для забезпечення стабільності при зростанні навантаження, необхідна оптимізація та масштабування через впровадження складніших стратегій кешування, оптимізацію запитів до бази даних та розгляд можливості переходу до асинхронної обробки завдань з використанням інструментів на кшталт Celery. Ці напрямки вказують на значний потенціал для росту та вдосконалення проєкту.

ВИСНОВОК ДО РОЗДІЛУ 4

Проведений аналіз розробленого вебзастосунку для бронювання квитків підтвердив успішну реалізацію основного функціоналу та досягнення ключових цілей дипломного проєкту. Створена система ефективно забезпечує перегляд подій, пошук, деталізацію інформації та безпечну купівлю квитків через Stripe, а також повноцінну реєстрацію та авторизацію користувачів.

Оцінка користувацького інтерфейсу показала його достатню простоту та інтуїтивність для виконання основних завдань. Обраний технологічний стек — Python, Django, PostgreSQL, Docker — виявився адекватним для проєкту такого обсягу, забезпечивши як швидкість розробки, так і стабільність основного функціоналу.

Водночас, виявлено обмеження, зокрема у гнучкості управління типами подій та квитків, що вказує на значний потенціал для майбутнього вдосконалення. Таким чином, реалізований проєкт є міцною функціональною основою, здатною до подальшого розвитку у більш складну та багатофункціональну платформу, що підтверджує ефективність обраних підходів та технологій.

ВИСНОВКИ

Ця дипломна робота стала результатом глибокого дослідження та практичної розробки вебзастосунку для продажу квитків, що має на меті оптимізувати процес купівлі та управління квитками, надаючи користувачам інтуїтивно зрозумілий і безпечний інструмент. Протягом роботи було всебічно проаналізовано існуючі рішення на ринку, ідентифіковано їхні переваги та недоліки, що дозволило сформувати чітке бачення функціональних і нефункціональних вимог до розроблюваної системи.

Основні досягнення та результати

В процесі виконання дипломної роботи було успішно досягнуто поставлених цілей. Було спроектовано та реалізовано повноцінний вебзастосунок, який відповідає сучасним вимогам до онлайн-сервісів. Ключові досягнення включають:

Розробка функціоналу для користувачів: Створено зручний інтерфейс для пошуку подій, перегляду детальної інформації про них, вибору місць на інтерактивній схемі зали та швидкої оплати квитків. Інтегровано механізми реєстрації та авторизації.

Реалізація адміністративної панелі: Розроблено інструменти для ефективного управління подіями, категоріями, користувачами та замовленнями. Адміністратор системи може додавати, редагувати та видаляти події, встановлювати ціни, керувати доступністю місць, що забезпечує гнучкість та оперативність в адмініструванні платформи.

Забезпечення безпеки та надійності: Особливу увагу було приділено питанням безпеки даних. Реалізовано захист від SQL-ін'єкцій, XSS-атак та інших поширених вразливостей. Застосовані сучасні методи шифрування для

зберігання конфіденційної інформації, що гарантує захист персональних даних користувачів та їхніх платіжних операцій.

Використання сучасних технологій: Для розробки застосунку було обрано перевірені та актуальні технологічні рішення, що забезпечують високу продуктивність, масштабованість та простоту подальшої підтримки. Вибір конкретних технологій обґрунтовано в аналітичній частині роботи, що підкреслює їхню доцільність для даного проекту.

Тестування та валідація: Здійснено всебічне тестування всіх модулів системи на предмет функціональності, надійності та відповідності вимогам. Проведені тести підтвердили працездатність застосунку та його готовність до експлуатації в реальних умовах.

Аналіз та обговорення результатів

Розроблений вебзастосунок демонструє високий потенціал для комерційного використання. Він не тільки забезпечує базові функції продажу квитків, але й пропонує додаткові можливості, які виділяють його серед конкурентів. Інтерактивна схема вибору місць, гнучкі налаштування подій для адміністратора та адаптивний дизайн роблять систему зручною та ефективною як для організаторів подій, так і для відвідувачів.

Порівняльний аналіз з існуючими аналогами показав, що застосунок має низку переваг, зокрема, у простоті користувацького інтерфейсу та гнучкості адміністрування. Хоча деякі комерційні платформи можуть мати ширший спектр інтеграцій (наприклад, з різними CRM-системами), проект є міцною основою, яку можна легко розширювати та адаптувати під конкретні потреби.

Перспективи подальшого розвитку

Незважаючи на успішне завершення поточної фази розробки, потенціал для подальшого вдосконалення залишається значним. Перспективні напрямки включають впровадження системи сповіщень про майбутні події та зміни, розробку розширених аналітичних інструментів для адміністраторів, створення нативних мобільних додатків, інтеграцію системи лояльності та розробку модуля персоналізованих рекомендацій на основі поведінки користувачів.

Підсумовуючи, виконаний дипломний проєкт підтверджує отримані теоретичні знання та практичні навички у сфері проектування та розробки вебзастосунків. Створений вебзастосунок для продажу квитків є повноцінним, безпечним та функціональним рішенням, яке може бути успішно впроваджене та використане в реальних умовах. Цей проект не тільки є вагомим внеском у власний професійний розвиток, але й демонструє можливість вирішення актуальних задач сучасної ІТ-індустрії. Він слугує міцним фундаментом для подальших наукових досліджень та практичних розробок у сфері електронної комерції та онлайн-сервісів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/
2. Django Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.djangoproject.com/en/stable/
3. PostgreSQL Official Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/current/
4. Git Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://git-scm.com/doc
5. GitHub Actions Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.github.com/en/actions
6. JavaScript Guide – MDN Web Docs [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
7. Django Template Language [Електронний ресурс] – Режим доступу до ресурсу: https://docs.djangoproject.com/en/stable/ref/templates/language/
8. Docker Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.docker.com/
9. Clean Architecture – WhatIs.com [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/whatis/definition/clean-architecture
10. OWASP. Top 10 Web Application Security Risks [Електронний ресурс] – Режим доступу до ресурсу: https://owasp.org/www-project-top-ten/
11. Django Security Overview [Електронний ресурс] – Режим доступу до ресурсу: https://docs.djangoproject.com/en/stable/topics/security/
12. MDN Web Docs – HTML Forms Guide [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Forms

13. JSON and APIs in Python [Електронний ресурс] – Режим доступу до ресурсу: https://realpython.com/python-json/
14. FreeCodeCamp. Git та GitHub: швидкий курс [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/git-and-github-crash-course/
15. MDN. Семантична розмітка HTML [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Glossary/Semantics
16. W3Schools. JavaScript Events [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/js/js_events.asp

ДОДАТОК А

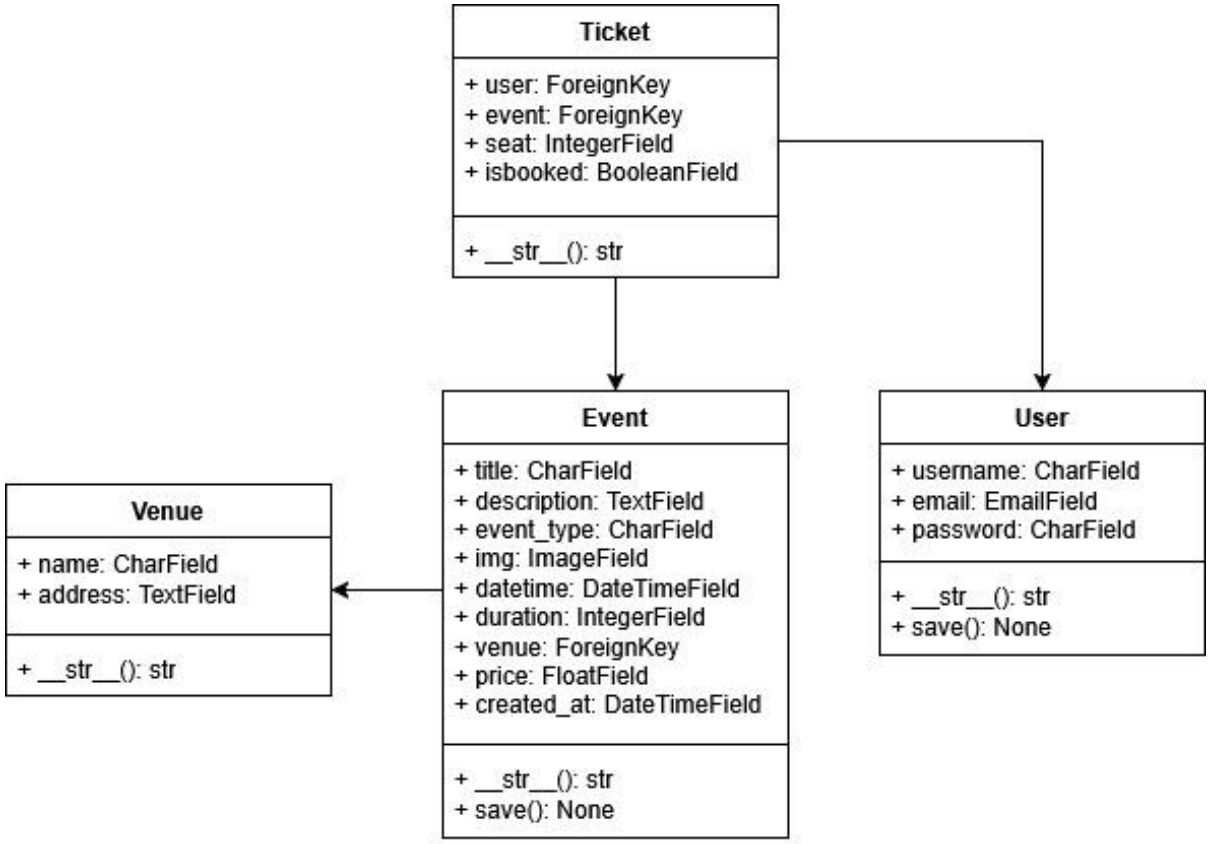
Веб-застосунок для бронювання квитків

Схема Django моделей

ІАЛЦ.467200.004 ДА

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.004 ДА		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Риндя І. А.			**Веб-застосунок для бронювання квитків** Схема Django моделей	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Каплунов А. В.					
Реценз.		Тимофєєва Ю.С.				КПІ ім. Ігоря Сікорського, ФІОТ, ІП-73	
Н. Контр.		Нікольський С.С.					
Затвердив							

ДОДАТОК Б

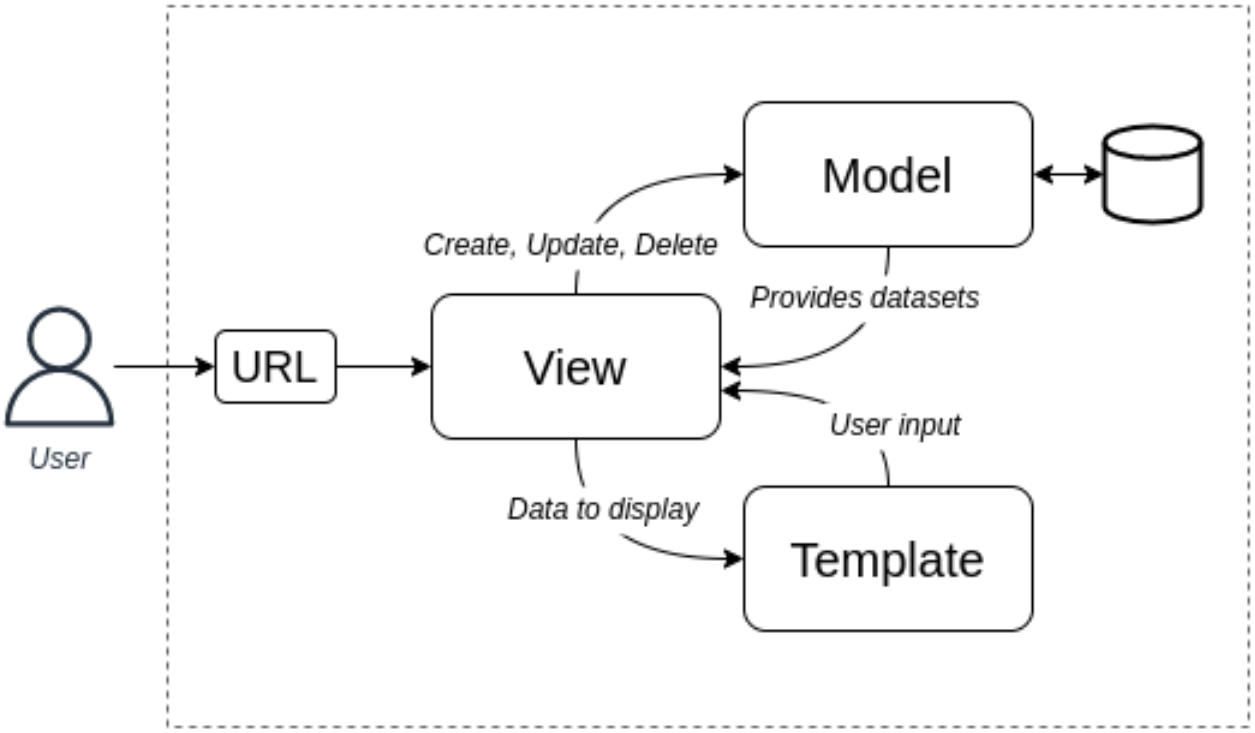
Веб-застосунок для бронювання квитків

Структурна схема MVT патерну в Django

ІАЛЦ.467200.005 ДБ

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.005 ДБ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Риндя І. А.			**Веб-застосунок для бронювання квитків** Структурна схема MVT патерну в Django	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Каплунов А. В.					
Реценз.		Тимофєєва Ю.С.				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-14	
Н. Контр.		Нікольський С.С.					
Затвердив							

ДОДАТОК В

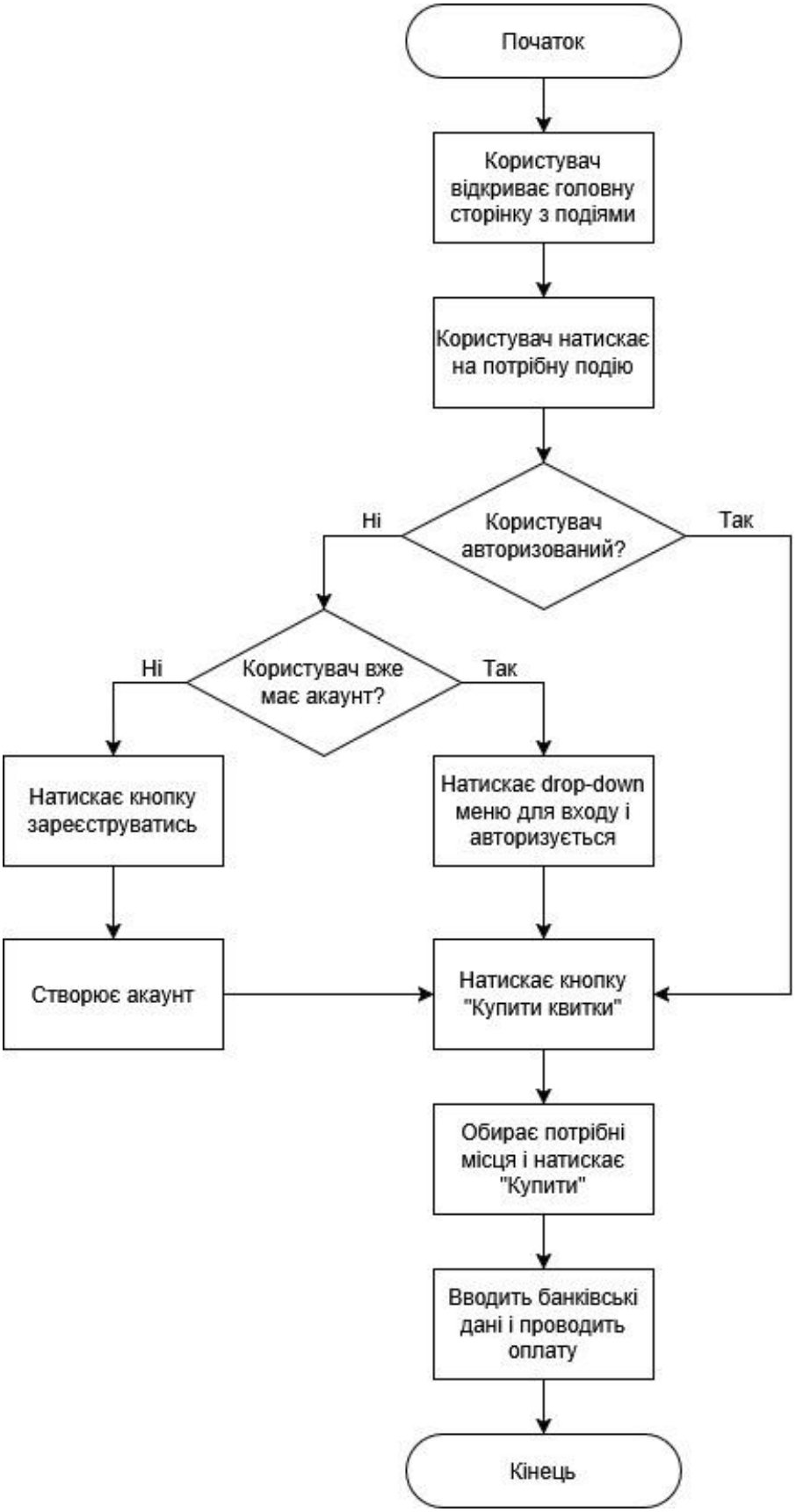
Веб-застосунок для бронювання квитків

Алгоритм дій веб-застосунку

ІАЛЦ.467200.006 ДВ

Аркушів 1

Київ 2025 р



					ІАЛЦ.467200.006 ДВ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Риндя І. А.			**Веб-застосунок для бронювання квитків** Алгоритм дій веб-застосунку	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Каплунов А. В.					
Реценз.		Тимофєєва Ю.С.				КПІ ім. Ігоря Сікорського, ФІОТ, ІО-14	
Н. Контр.		Нікольський С.С.					
Затвердив							

ДОДАТОК Г

Веб-застосунок для бронювання квитків

Текст програмного коду

ІАЛЦ.467200.007 ДГ

Аркушів 9

Київ 2025 р

```
from django.db import models
from django.utils.timezone import now
from django.contrib.auth.models import User

def upload_to(instance: 'Event', filename):
    created_at = instance.created_at or now()
    return f"uploads/images/{created_at.strftime('%Y/%m/%d')}/{filename}"

class Venue(models.Model):
    name = models.CharField(max_length=100, verbose_name='Назва локації')
    address = models.TextField(verbose_name='Адреса')

    def __str__(self):
        return self.name

class Event(models.Model):
    EVENT_TYPES = [
        ('cinema', 'Кіно'),
        ('theatre', 'Театр'),
        ('concert', 'Концерт'),
        ('exhibition', 'Виставка'),
    ]
    title = models.CharField(max_length=100, verbose_name='Назва')
    description = models.TextField(verbose_name='Опис', blank=True, null=True)
    event_type = models.CharField(max_length=20, choices=EVENT_TYPES, verbose_name='Тип
події')
    img = models.ImageField(upload_to=upload_to, verbose_name='Посилання на
зображення', blank=True, null=True)
    datetime = models.DateTimeField(verbose_name='Дата та час')
    duration = models.IntegerField(verbose_name='Тривалість (хв)', default=0)
    venue = models.ForeignKey(Venue, on_delete=models.CASCADE, verbose_name='Локація')
    price = models.FloatField(verbose_name='Ціна')
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.title} ({self.get_event_type_display()})"

    def save(self, *args, **kwargs):
        super().save(*args, **kwargs)
        if not self.tickets.exists():
            for seat in range(90):
                Ticket.objects.get_or_create(event=self, seat=seat)

class Ticket(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, verbose_name='Користувач',
related_name='tickets', null=True, blank=True)
    event = models.ForeignKey(Event, on_delete=models.CASCADE, verbose_name='Подія',
related_name='tickets')
    seat = models.IntegerField(verbose_name='Місце')
    isbooked = models.BooleanField(verbose_name='Заброньовано', default=False)

    def __str__(self):
        return f'{self.event} — місце №{self.seat}'
```

```
from django.shortcuts import redirect, render
from django.views.generic import DetailView, ListView
from django.http import HttpResponseRedirect, JsonResponse
from .models import *
from django.views.decorators.csrf import csrf_protect
from django.contrib import messages
from django.contrib.auth import login, logout, authenticate
from django.contrib.auth.models import User
# Create your views here.

class EventListView(ListView):
    model = Event
    template_name = 'home.html'
    paginate_by = 30

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["title"] = 'KyivTickets: Розклад сеансів та квитки онлайн у центрі
Києва'
        context["allevents"] = Event.objects.all()
        return context

def contacts(request):
    context = {'title': 'Контакти - KyivTickets'}
    context["allevents"] = Event.objects.all()
    return render(request, 'contacts.html', context)

def about(request):
    context = {}
    context["title"] = 'Про нас - KyivTickets: Наша історія та місія'
    context["allevents"] = Event.objects.all()
    return render(request, 'about.html', context)

class EventDetailView(DetailView):
    model = Event
    template_name = "event.html"
    context_object_name = "event"

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["title"] = f'{Event.objects.get(pk=self.kwargs["pk"]).title} -
KyivTickets'
        context["allevents"] = Event.objects.all()
        return context

def user_signup(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        email = request.POST.get('email')
        password1 = request.POST.get('password1')
        password2 = request.POST.get('password2')
        if not username or not email or not password1 or not password2:
            messages.error(request, 'Заповніть всі поля')
            return render(request, 'signup.html')
        if password1 != password2:
            messages.error(request, 'Паролі не співпадають')
            return render(request, 'signup.html')
        user = authenticate(username=username, password=password1)
        if user:
            messages.error(request, 'Користувач з таким іменем вже існує')
            return render(request, 'signup.html')
        if User.objects.filter(email=email).first():
```

```
            messages.error(request, 'Користувач з такою поштою вже існує')
            return render(request, 'signup.html')
        user = User.objects.create_user(username=username, email=email,
password=password1)
        user.save()
        login(request, user)
        return redirect('home')
    context = {'title': 'Реєстрація - KyivTickets'}
    context["allevents"] = Event.objects.all()
    return render(request, 'signup.html', context)

def user_login(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        next = request.POST.get('next', '/')
        user = authenticate(username=username, password=password)
        if user:
            login(request, user)
        else:
            messages.error(request, 'Неправильний логін або пароль')
    return HttpResponseRedirect(next)

def user_logout(request):
    if request.method == 'POST':
        next = request.POST.get('next', '/')
        logout(request)
        return HttpResponseRedirect(next)


import stripe
from django.conf import settings
from django.shortcuts import get_object_or_404, redirect
from django.http import HttpRequest, JsonResponse
from django.views.decorators.csrf import csrf_exempt
from django.contrib.auth.decorators import login_required
from main.models import Event, Ticket

stripe.api_key = settings.STRIPE_SECRET_KEY

@csrf_exempt
@login_required
def create_checkout_session(request: HttpRequest):
    if request.method != 'POST':
        return JsonResponse({'error': 'Invalid method'}, status=405)

    import json
    data = json.loads(request.body)
    ticket_ids = data.get('tickets', [])

    if not ticket_ids:
        return JsonResponse({'error': 'No tickets provided'}, status=400)

    tickets = Ticket.objects.filter(id__in=ticket_ids, isbooked=False)

    if not tickets.exists():
        return JsonResponse({'error': 'No valid tickets'}, status=400)

    line_items = []
    metadata = {}

    for ticket in tickets:
```

```
        line_items.append({
            'price_data': {
                'currency': 'UAH',
                'product_data': {
                    'name': f'{ticket.event.title}',
                    'description': f'Seat #{ticket.seat}'
                },
                'unit_amount': int(ticket.event.price * 100)
            },
            'quantity': 1
        })
        metadata[str(ticket.id)] = request.user.id

    checkout_session = stripe.checkout.Session.create(
        success_url=f'http://{settings.APP_DOMAIN}/event/{tickets[0].event.id}',
        cancel_url=f'http://{settings.APP_DOMAIN}/event/{tickets[0].event.id}/cancel/',
        metadata={
            'ticket_ids': ','.join([str(t.id) for t in tickets]),
            'user_id': request.user.id
        },
        payment_method_types=['card'],
        mode='payment',
        line_items=line_items,
    )

    return JsonResponse({'url': checkout_session.url})

@csrf_exempt
def stripe_webhook(request: HttpRequest):
    payload = request.body
    sig_header = request.headers.get('Stripe-Signature', '')

    try:
        stripe_event = stripe.Webhook.construct_event(
            payload, sig_header, settings.STRIPE_WEBHOOK_SECRET
        )
    except ValueError:
        return JsonResponse({'error': 'Invalid payload'}, status=400)
    except stripe.error.SignatureVerificationError:
        return JsonResponse({'error': 'Invalid signature'}, status=400)

    if stripe_event['type'] == 'checkout.session.completed':
        session = stripe_event['data']['object']
        ticket_ids = session['metadata']['ticket_ids'].split(',')
        user_id = session['metadata']['user_id']

        for ticket_id in ticket_ids:
            ticket = Ticket.objects.filter(id=ticket_id, isbooked=False).first()
            if ticket:
                ticket.user_id = user_id
                ticket.isbooked = True
                ticket.save()

    return JsonResponse({'status': 'success'})


"""
Django settings for KyivCinema project.

Generated by 'django-admin startproject' using Django 4.2.5.

For more information on this file, see
```

```
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""

import os
from dotenv import load_dotenv
from pathlib import Path

load_dotenv()

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = os.getenv('DJANGO_SECRET_KEY')

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

APP_HOST = os.environ.get('APP_HOST')
APP_DOMAIN = APP_HOST if APP_HOST else 'localhost:8000'
ALLOWED_HOSTS = ['localhost', '127.0.0.1', '0.0.0.0'] + [APP_HOST] if APP_HOST else []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'main.apps.MainConfig',
    'corsheaders',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'corsheaders.middleware.CorsMiddleware'
]

CORS_ALLOW_ORIGINS = [
    "http://localhost:8000",
]

CORS_ALLOW_ALL_ORIGINS = True

ROOT_URLCONF = 'ticket_project.urls'

TEMPLATES = [
    {
```

```
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'ticket_project.wsgi.application'

# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.environ.get('DB_NAME', 'postgres'),
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': os.environ.get('DB_HOST', 'localhost'),
        'PORT': '5432',
    }
}

# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/
```

```
STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')



import os
import django

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'ticket_project.settings')
django.setup()

from django.contrib.auth import get_user_model

User = get_user_model()

def get_env_var(var_name):
    value = os.getenv(var_name)
    if value is None:
        raise ValueError(f'Error: Missing required environment variable "{var_name}"')
    return value

def create_superuser():
    username = get_env_var('DJANGO_SUPERUSER_USERNAME')
    email = get_env_var('DJANGO_SUPERUSER_EMAIL')
    password = get_env_var('DJANGO_SUPERUSER_PASSWORD')

    if not User.objects.filter(username=username).exists():
        User.objects.create_superuser(username=username, email=email,
password=password)
        print('Superuser created successfully.')
    else:
        print('Superuser already exists.')

if __name__ == '__main__':
    create_superuser()



FROM python:3.12-alpine

ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1

WORKDIR /app

RUN pip install --upgrade pip && pip install poetry==2.1.2

COPY . /app
RUN poetry config virtualenvs.create false \
    && poetry install --no-interaction
```

```
RUN chmod +x ./docker-entrypoint.sh

EXPOSE 8000

ENTRYPOINT ["./docker-entrypoint.sh"]
```

```
services:
  web:
    build: .
    container_name: django_app
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - .:/app
    ports:
      - "8000:8000"
    depends_on:
      - db
    environment:
      - DB_HOST=db

  db:
    container_name: postgres_db
    image: postgres:15
    restart: always
    environment:
      POSTGRES_DB: postgres
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 10s
      timeout: 5s
      retries: 5
  stripe-cli:
    container_name: stripe-cli
    image: stripe/stripe-cli
    command: listen --api-key ${STRIPE_SECRET_KEY} --device-name 'docker-cli' --
forward-to http://host.docker.internal:8000/payments/webhook/
    env_file:
      - .env

volumes:
  postgres_data:
```

```
name: Python style check

on: [push, pull_request]

jobs:
  style:
    runs-on: ubuntu-latest

    steps:
```

```
    - uses: actions/checkout@v2

    - name: Set up Python 3.11
      uses: actions/setup-python@v2
      with:
        python-version: 3.11

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install flake8

    - name: Run flake8
      run: flake8 .
      continue-on-error: false
```

```
#!/bin/sh

# Apply database migrations
echo "Applying database migrations..."
python manage.py migrate

# Wait for PostgreSQL to be ready
echo "Waiting for PostgreSQL..."
while ! nc -z db 5432; do
  sleep 0.1
done
echo "PostgreSQL is up!"

# Create a superuser if it doesn't exist
python /app/create_superuser.py

# Start Django application
echo "Starting Django server..."
exec "$@"
```