МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ Олександр ЛИТВИНЕНКО
"_____" _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)
ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Програмна система на основі блокчейну для купівлі авіаквитків

Виконавець: Владислав КОШЕЛЕНКО

Керівник: Надія МАРЧЕНКО

Нормоконтролер: Євгеній ТУПОТА

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп’ютерних наук та технологій

Кафедра комп’ютеризованих систем управління

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Системне програмування»

Форма навчання денна

ЗАТВЕРДЖУЮ

Завідувач кафедри

Олександр ЛИТВИНЕНКО

“____” _____2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Кошеленко Владислава Валерійовича

1. Тема кваліфікаційної роботи Програмна система на основі блокчейну для купівлі авіаквитків

затверджена наказом ректора від «_28_» __08__2023 р. №1494/ст

2. Термін виконання роботи: з 02.10.2023 по 31.12.2023

3. Вихідні дані до роботи: мова програмування *Python*, середовище розробки *Visual Studio Code*, блочейн з алгоритмом консенсусу *PoS*, швидкість транзакцій.

4. Зміст пояснювальної записки: вступ, аналіз предметної області, використання програмних засобів та бібліотеки, опис реалізації завдання, приклади застосування, висновки.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) Схема алгоритму консенсусу *POS*;
2) класи програмної системи;
3) елементи згенерованого *UUID*;
4) використання *Flask* для взаємодії з блокчейном;
5) процес генерації підпису;
6) типовий потік запитів.

6. Календарний план

№ п/п	Етапи виконання кваліфікаційної роботи	Термін виконання етапів	Примітка
1	Пошук та аналіз джерел для аналізу предметної області за темою кваліфікаційної роботи	02.10-22.10	
2	Розробка плану дипломного проекту	23.10-03.10	
3	Розробка розділу 1: Аналіз предметної області	04.11-14.11	
4	Розробка розділу 2: Використання програмних засобів та бібліотеки	15.11-25.11	
5	Розробка розділу 3: Опис реалізації завдання	26.11-03.12	
6	Розробка розділу 4: Приклади застосування та підготовка до попереднього захисту	04.12-11.12	
7	Оформлення пояснювальної записки, написання висновків	12.12–19.12	
8	Розробка презентації для захисту дипломного проекту	20.12–25.12	
9	Підговка до захисту	26.12–31.12	

7. Дата отримання завдання «02» 10. 2023 р.

Керівник дипломного проекту _____ Надія МАРЧЕНКО

Завдання прийняв до виконання _____ Владислав КОШЕЛЕНКО

РЕФЕРАТ

Кваліфікаційна робота на тему «Програмна система на основі блокчейну для купівлі авіаквитків». Записка до кваліфікаційної роботи містить: 91 с., 35 рис., 23 літературних джерела.

Ключові слова: БЛОКЧЕЙН, СИСТЕМА, КВИТКИ, АВІАЦІЯ, *POS*.

Об’єкт – процес придбання авіаквитків.

Предмет – програмна система на основі блокчейну для купівлі авіаквитків.

Метою кваліфікаційної роботи є створення програмної системи з вебінтерфейсом, яка використовує технологію блокчейн для придбання авіаквитків.

Методи: аналіз та порівняння існуючих прикладів використання технологій блокчейну для купівлі авіаквитків, проектування та розробка блокчейну, створення веб-інтерфейсу та тестування

Технічні засоби:

1) мови програмування - *Python, HTML, CSS*;
2) бібліотеки, фреймворки та інші технології, необхідні для розробки та реалізації системи - *Uuid, P2pnetwork, Threading, Flask_classful ma Flask, Copy, Sys, Requests, Json ma Jsonpickle, Cryptodome*;
3) програмні засоби: середовище розробки, *IDE*, редактори коду та інші інструменти - *Visual Studio.*

В результаті отримано окремий блокчейн з алгоритмом консенсусу *PoS* та додатковим класом який містить самі авіаквитки, в середині кожного блоку. Він більше не вимагає великих апаратних ресурсів та немає комісій - це знизить ціни на квитки, спростить та пришвидшить процес купівлі. Подібні системи ще знаходяться на стадії розробки, тому моя програмна система немає аналогів.

ЗМІСТ

ВСТУП

Після перемоги у війні Україні потрібно повертатися до нормального життя та розвиватись. Для прискорення цього процесу потрібно використовувати сучасні технології та створювати власні нові. Одна з найбільш популярних технологій зараз це блокчейн. Його намагаються інтегрувати розвинуті країни та компанії в різні сфери, оскільки це важлива технологія яка до того ж добре взаємодіє з іншими проривними технологіям, такими як: штучний інтелект, інтернет речей та великі даними.

Якщо використання блокчейну в сфері криптовалют, фінансів або державного управління вже не викликає здивування, то використання цієї технології для реалізації квитків поки на стадії розробки. Мішель Кадо, міжвідомчий делегат французького уряду, відповідальний за підготовку до Олімпійських ігор-2024, пропонує продавати квитки на інтегрувавши в процес блокчейн. Ці квитки буде неможливо передати чи перепродати іншим людям, тобто проблема спекуляції може бути успішно вирішена. Усі квитки автоматично будуть згенеровані та відправлені покупцям за кілька днів до Олімпіади. Ця інновація також може бути протестована на чемпіонаті світу з регбі 2023 року та Міжнародному турнірі з тенісу. Обговорення впровадження блокчейн-технологій активізувалося після дуже невдалого процесу допуску до фінального матчу Ліги Чемпіонів між мадридським "Реалом" та англійським "Ліверпулем" 28 травня цього року, коли подія була відкладена на 30 хвилин у зв'язку з напливом вболівальників без квитків. Попередній досвід використання цифрових технологій під час Олімпіади-2022 в Пекіні підтвердив значний інтерес користувачів до цієї сфери. Зокрема, цифровий юань активно використовувався тисячами відвідувачів, і кількість транзакцій з його використанням перевищила традиційні платежі через систему *VISA* [3]. Переваги блокчейну для продажу квитків:

1) блокчейн дозволяє встановити повний контроль над продажем та використанням квитків, що ефективно допомагає мінімізувати організаційні та корупційні ризики;
2) проблему спекуляції та перепродажу квитків можна вирішити, оскільки всі транзакції контролюються та відображаються в блокчейні;
3) прискорення всіх транзакцій та мінімізація можливих технічних проблем на всіх етапах реалізації програми;
4) позитивний вплив на імідж компанії та всіх основних спонсорів завдяки більшій залученості відвідувачів (особливо серед молоді).

У випадку успішної реалізації цієї ініціативи поширеність блокчейн-технологій для купівлі квитків може зрости. Крім того, вони не обов'язково повинні бути реалізовані за однієї моделлю. Зокрема, перепродаж квитків може бути дозволено і це також може бути реалізовано в блокчейні. Можливо встановити максимальну надбавку до вартості покупки або обмеження на кількість перепродажів. Можуть бути впроваджені додаткові вимоги до отримувачів: наприклад, за географічним ознакою, належністю до певної організації або профспілки. Іншими словами, блокчейн дозволяє реалізувати практично будь-яку схему розповсюдження квитків - в залежності від планів та вподобань компанії. Головне, що будь-які вимоги можуть бути ефективно відображені в алгоритмах без додаткових витрат.

Блокчейн також надає вичерпну інформацію про всі транзакції, дозволяючи менеджерам та адміністраторам провести об'єктивний аналіз з виявленням сфер для подальшого удосконалення своїх послуг та схеми взаємодії з партнерами. При цьому рівень конфіденційності користувачів може варіюватися від максимальної анонімності чи псевдонімності до повної ідентифікації. Блокчейн дозволяє ставити різноманітні цілі, включаючи максимізацію прибутку і доходу, рівномірне представництво серед членів різних груп або інші економічні і суспільно важливі пріоритети. Блокчейн-рішення для продажу квитків можуть бути успішно інтегровані з іншими інноваціями, сприяючи технологічній еволюції цілих галузей в найближчі роки. У багатьох галузях блокчейн-технологія на сьогоднішній день ще

не знайшла свого застосування. Проте це не стосується галузі продажу квитків. В процесі розробки токенізації подій стало зрозуміло, що одним з використань токена є сам квиток. Завдяки цьому використанню квиток стає неможливим підробити чи скопіювати. Крім цього, всі взаємовідносини між організатором та покупцем регулюються блокчейном, що виключає невиконання домовленостей з боку організатора та захищає покупця уже не тільки в юридичному полі, а на рівні запрограмованого алгоритму. З приходом інтернету паперовий квиток став електронним. Тепер час для наступного еволюційного стрибка - перехід до крипто-рішень. Зміна парадигми затроне всі чотири цільові групи - користувачів, систем продажу квитків, організаторів, поширювачів. Користувач може бути впевненим, що квиток, який він купив на первинному або вторинному ринку, не підробка або дублікат. Покупець також може впливати на ринок через систему голосування. Наприклад, якщо на компанію багато скарг, то може бути запущено автоматичне голосування щодо якості обслуговування, і у разі невиконання передбачених послуг компанією гроші автоматично повернуться користувачам. Користувачі самі формують рейтинг систем продажу квитків і організаторів, що дозволяє системі швидко відсіювати та виключати недобросовісних гравців.

З свого боку компанії зможуть випускати криптоквитки, що захищають клієнтів. Технологія криптоквитків дозволяє як отримувати прибуток від додаткового продажу брокера вторинного ринку, так і навпаки, повністю заборонити перепродаж та передачу квитка від користувача до користувача. Можливе встановлення обмеження "один квиток в одні руки", можливість покупки тільки персоналізованими покупцями - така настройка повністю виключить покупку квитків ботами. Таким чином, процес випуску квитків стає повністю керованим та знаходиться в руках організатора. Організатори та поширювачі можуть взаємодіяти з глядачем через *push*-сповіщення та бонусні програми з нарахуванням токенів.

Авіаційна промисловість є однією з найбільших і найважливіших галузей економіки України, яка до того ж постраждала від війни чи не найбільше. Вона забезпечує зв'язок між різними регіонами країни та світу, а також є важливим джерелом бюджетних надходжень. В умовах післявоєнного відновлення України,

використання технології блокчейн авіакомпаніями є одним із ключових факторів для швидкого повернення на світовий ринок та підвищення їх конкурентоспроможності. Вона має потенціал революціонізувати авіаційну промисловість. Її основна особливість полягає в тому, що вона є надійною та недоступною для втручання. Саме тому вона допомагає забезпечити точність та недоторканість даних, що є надзвичайно важливим для авіаційної галузі. Саме через це авіаційна галузь, яка сильно покладається на цілісність даних, може скористатися перевагами останніх тенденцій у галузі блокчейн технологій. Простими словами, технологія блокчейн у сфері авіації надає спрощену систему для швидкого розподілу прибутку, зменшення загальних витрат та усунення можливих затримок через конфлікти щодо розподілу прибутку. Створюючи єдину систему, яка використовує визначені критерії, технологія блокчейн дозволяє забезпечити пунктуальність рейсів та уникнути витратних затримок. Отже, блокчейн в авіакомпаніях може значно сприяти керівникам повітряних судів, авіакомпаніям та пасажирам.

Узагальнюючи, завдяки потенціалу надання реальних користей та безшовного подорожнього досвіду, технологія блокчейн набуває все більшої популярності в авіаційній галузі.

Безумовно, технологія блокчейн - наступна велика річ у секторі авіації. За даними *Markets and Markets*, ринок блокчейну у авіакомпаніях прогнозується зрости з 421 мільйонів доларів у 2019 році до 1394 мільйонів доларів до 2025 року, з річним темпом зростання на рівні 22,1% протягом прогнозованого періоду. Ринок авіаційного блокчейну у Північній Америці прогнозується зрости на 25,2%, що є найвищим темпом річного зростання протягом прогнозованого періоду. Північна Америка є найбільш розвиненою регіоном у впровадженні нових технологій та інфраструктури. Ймовірно, збільшення темпу впровадження блокчейну в повітряний транспорт є одним із значущих факторів, які впливають на темп росту цього регіону [4]. Технологія блокчейну має потенціал революціонізувати авіаційну промисловість. Вона допомагає покращити ефективність операцій, зменшити витрати та підвищити безпеку.

Отже, метою кваліфікаційної роботи є розробка програмної системи з вебінтерфейсом, яка використовує технологію блокчейн для придбання авіаквитків.

Для досягнення цієї мети необхідно вирішити такі завдання:

1) розглянути існуючі проблеми та провести аналіз різних блокчейн-рішень, які використовуються у сфері сфері купівлі авіаквитків;
2) проаналізувати подібні системи на основі блокчейну, які використовуються для купівлі авіаквитків;
3) обґрунтувати використання бібліотек та інструментів, які використовуються для створення системи;
4) розробити програмну систему;
5) описати варіанти використання системи.

Об’єктом є процес придбання авіаквитків. Предметом – програмна система на основі блокчейну для купівлі авіаквитків.

Вибрані методи дослідження дозволяють досягти поставленої мети.

Аналіз та порівняння існуючих прикладів використання технологій блокчейну для купівлі авіаквитків - цей метод дослідження є важливим, оскільки він дозволяє отримати уявлення про існуючі рішення та проблеми, пов'язані з використанням технології блокчейну для купівлі авіаквитків. Це допоможе розробити більш ефективне та зручне рішення. Для проведення аналізу та порівняння існуючих прикладів було пройдено такі кроки:

1) аналіз наукової літератури;
2) аналіз вебсайтів та додатків.

Проектування та розробка блокчейну - цей метод дослідження є важливим, оскільки він дозволяє створити блокчейн, який відповідає вимогам розробленої системи. Блокчейн повинен бути безпечним, масштабованим та швидким. Для проектування та розробки блокчейну було пройдено такі кроки:

1) аналіз вимог;
2) вибір архітектури;
3) розробка алгоритмів;
4) виконання тестування.

Створення вебінтерфейсу - цей метод дослідження є важливим, оскільки він дозволяє створити вебінтерфейс, який дозволяє користувачам взаємодіяти з блокчейном. Вебінтерфейс повинен бути простим у використанні та інтуїтивно зрозумілим. Для створення вебінтерфейсу було пройдено такі кроки:

1) аналіз вимог;
2) дизайн інтерфейсу;
3) розробка інтерфейсу;
4) виконання тестування.

Згідно останніх досліджень та розробок блокчейн може бути важливим інструментом для авіаційних компаній. Проте суттєвим недоліком попередніх розробок був алгоритм консенсусу - *PoW*, який вимагав чималих ресурсів. Для вирішення цієї проблеми створили новий алгоритм *PoS*, що використовується в моїй системі, оскільки це надало другий поштовх на розробку подібних систем. Проте зараз, на мою думку, проблема в том що компанії намагаються використовувати вже готові блокчейни, які перегружені через свою популярність в цілому, тому вони вже перейшли до створення своїх блокчейнів. Моя система є однією з перших такого застосування. Блокчейн працює безперебійно, на даному етапі транзакції проходять за 2-3 секунди, проте код можна оптимізувати більше. Комісії з транзакцій немає, що позитивно вплина на ціну. Квитки використовуються у виді окремих токенів. Система обирає кожну наступну ноду, яка зможе дістати блок за допомогою лотереї, є можливість відправити монети до окремого місця що збільшить шанс отримання можливості на видобуток нового блоку.

Отже, отримана система є конкурентно здібною враховуючи швидкість транзакцій на даному етапі та повну відсутність комісій. Використовувати блокчейн можна в будь-якій точні планети, що не вплине на швидкість чи комісію.

Розроблену програмну систему слід використовувати авіаційними компаніями з метою поліпшення процедури придбання авіаквитків, хоча можна взяти за основу для системи купівлі квитків в будь-якій галузі. Рішення може бути впроваджено на етапі пілотного проекту з подальшим масштабуванням.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Загальна інформація про використання технології блокчейн для купівлі авіаквитків

Якщо глибще поглибитись в основні характеристики та принципи блокчейну, можна виділити наступне:

1) децентралізований: блокчейн є децентралізованим, що означає відсутність управлінського органу чи однієї особи, яка б відповідала за структуру. Іншими словами, ви можете зберігати будь-що, починаючи від криптовалют до цінних цифрових активів та інших важливих документів, не залучаючи посередника чи навіть управлінського органу;
2) незмінність: незмінність вказує на те, що щось не може бути змінено чи піддано впливу. Оскільки технологія блокчейн недоступна для вторгнень, ніхто не може змінити систему, не повідомивши інших у мережі блокчейн;
3) підвищена безпека: використання шифрування в блокчейні надає системі дуже міцний шар безпеки. Блокчейн використовує криптографію (математичний алгоритм, який виступає як брандмауер проти атак), щоб додати ще один рівень захисту для користувачів. Кожен елемент інформації, присутній в блокчейні, криптографічно схований. Отже, можна сказати, що інформація в мережі приховує справжню природу даних.

Блокчейн має можливість повністю змінити авіаційну індустрію, наприклад деякі приклади його використання описані нижче:

1) збір та зберігання даних: найбільшою загадкою в авіаційній індустрії є зникнення рейсу Малайзійських авіаліній 370, яке сталося у березні 2014 року. Було недостатньо деталей про те, чому сталася дивна зміна шляху польоту та куди направлявся рейс. Мільйони доларів були витрачені на встановлення деталей рейсу *MH370*, але нічого не було досягнуто. Це

нещасне випадок є гарним прикладом того, що якби у нас була краща система нагляду на той час, ми могли б впоратися з викликом пошуку пропавшого літака. Блокчейн - сучасний тренд у сфері технологій. Він допомагає збирати та зберігати всі дані, пов'язані з одним літаком, і ці дані можна використовувати різними способами, такими як:

- запис місцезнаходження цінних активів в реальному часі та надійний спосіб. Інформація, така як траєкторія польоту, посадка багажу, виявлення втраченого об'єкта, деталі пасажирів тощо, може бути корисною в нещасних випадках;
- забезпечення всієї інформації, пов'язаної з польотом, віддалено на резервному записі блокчейну може бути додатковим заходом безпеки до сьогоднішніх централізованих систем, які вразливі до вторгнень. Це також забезпечить прозорість та спрощений процес;

2) економія витрат на обслуговування: ми всі мали досвід затримок рейсу в аеропорту через проблеми з технічним обслуговуванням. Ці затримки не тільки призводять до погіршення репутації бренду, але й призводять до втрати безлічі доларів. Ось як блокчейн у сфері авіації допомагає зекономити мільйони доларів на технічному обслуговуванні:

- блокчейн може постійно оновлювати журнали стану кожної частини літака. Це допоможе скоротити час, витрачений на регулярний огляд та технічне обслуговування літака;
- наука за технологією реєстру розвинена до такого вражаючого ступеня, що постійне оновлення реєстра покращить ефективність та підвищить використання;
- блокчейн також може допомогти в прогнозуванні технічного обслуговування та ліквідації проблем до того, як вони вплинуть на необхідну авіакомпанійною діяльність;
- постачальники *MRO* також можуть використовувати блокчейн та надавати перевірену документацію для компонентів, які вони обслуговували чи встановлювали;

3) впроваджує концепцію «розумних квитків»: теоретична концепція "розумних квитків" може бути втілена за допомогою технології блокчейн. Це не тільки усуває використання паперових квитків, але також може бути використане для отримання доступу до аеропортового лаунжу, проживання тощо. Додатково, зберігання "розумних квитків" на блокчейні може допомогти пом'якшити або усунути ефекти хаосу, які можуть призвести до збоїв у централізованій базі даних квитків авіакомпанії чи весь аеропорт. Така токенізація активів також дозволить еволюціонувати систему бронювання, усуваючи потребу в альтернативних формах ідентифікації;
4) підтверджує ідентифікацію користувачів: авіаційна галузь стикається з серйозними ризиками через підробку особистостей, включаючи ризик терористичних актів на борту літаків чи в аеропортах. Для вирішення цього питання компанії можуть використовувати технологію блокчейн для підтвердження особистостей біометричними засобами. Коли ідентифікація підтверджена, вона реєструється в блокчейні, який є децентралізованим і надійним, що означає, що її неможливо змінити. Це не тільки зменшує людські помилки під час процесу перевірки документів, але й, в кінцевому підсумку, призводить до того, що біометрична ідентифікація замінить паперові паспорти. Таким чином, технологія блокчейн може надати надійний та безпечний спосіб підтвердження особистостей в авіаційній промисловості;
5) покращує цифровий досвід подорожування: нефективність - це витрата, яку всю галузь туризму мусить нести. Оскільки споживачі люблять заощадити трохи тут і там, компанії, які організовують бронювання подорожей, можуть бути відповідальні за суми до $269,000 на витрати. Технологія блокчейн в авіаційній промисловості має можливість об'єднати системи для різних галузей, таких як авіакомпанії та інші галузі туризму, такі як продаж квитків, програми лояльності та не-авіаційні логістичні галузі, такі як транспорт та готелі. Усі галузі, пов'язані з подорожами, можуть

використовувати уніфікацію, щоб створити безшовний досвід подорожей та покращити задоволення клієнтів.

Подальший розвиток технології блокчейн та застосування його в авіаційній індустрії призведе до поширення використання подібних систем та їх взаємодії. Велика потенційна користь у впровадженні блокчейну у галузі купівлі авіаквитків може стати стимулом для активного розвитку та вдосконалення цих технологій у майбутньому[5].

1.1.1. Користь блокчейну для авіаційного сектору

Традиційна авіаційна промисловість має справу з застарілою та відокремленою системою, що ускладнює швидкий та безшовний обмін інформацією між складними мережами учасників галузі. Саме тут технологія блокчейн в авіації виступає як великий рятувальник. Ця технологія пропонує децентралізоване рішення для цих проблем. Завдяки своїм можливостям обміну, блокчейн може кардинально покращити ефективність, доступність та адаптивність, при цьому знижуючи витрати та дозволяючи впроваджувати нові бізнес-моделі. Це робить блокчейн ідеальним рішенням для існуючих проблем у авіаційній промисловості.

Авіаційна галузь є високорегульованою, проте їй також потрібна радикальна оптимізація. Технологія блокчейн - це рішення для революції в авіаційній індустрії та створення величезних економій для авіакомпаній. Комбінуючи блокчейн з іншими технологіями, такими як інтернет речей та штучний інтелект, авіакомпанії можуть розумно переосмислити, як вони ведуть бізнес, і отримати найбільші вигоди.

Оцінюється, що ринок авіаційного блокчейну зросте приблизно на 22% від 2019 до 2025 року [6]. В авіації зростає інтерес до технології блокчейну для забезпечення можливостей відстеження та прозорості в операціях, зменшення складності ланцюга постачання, цифровізації бізнес-процесів та покращення комунікацій та взаємодії з пасажирами. Можливість відстеження в блокчейні підтримує авіаційну промисловість на всіх етапах ланцюга постачання для

забезпечення більш сталих практик. Шляхом відстеження складного ланцюга постачання вгору, промисловість може отримати більш детальні відомості про джерела та складові матеріалів і компонентів, сертифіковану кількість вторинної сировини в деталях чи біорозчинних компонентів у сталих авіаційних паливах, а також екологічні впливи кожного етапу процесу виробництва. Це дозволяє всім учасникам ланцюга постачання - від постачальників сировини до виробників деталей, *OEM* та авіакомпаній - приймати більш сталі рішення щодо дизайну екстер'єру та інтер'єру в кабіні повітряного судна для зменшення відходів та викидів. В результаті галузь зможе оптимізувати свій ланцюг постачання, зменшити складність та ризик у виробничому процесі. Відстеження ланцюга постачання за допомогою блокчейну також може підтримувати діяльність в області технічного обслуговування, ремонту та операцій (*MRO*) в авіаційній промисловості. Воно створює запис про склад і джерело деталей та компонентів вгору у формі цифрових паспортів продукту і може включати такі елементи, як інструкції з розбирання. В результаті компанії можуть продовжувати терміни служби продуктів та відстежувати друге чи третє життя деталей та компонентів, що дозволяє в майбутньому використовувати більш циркулярні бізнес-моделі. Що ще важливо, відстеження ланцюга постачання за допомогою блокчейну дозволяє компаніям в авіаційній промисловості краще комунікувати свої дії споживачам. Наприклад, пасажири авіакомпаній можуть сканувати *QR*-коди на своїх місцях та інших ділянках кабіни, щоб побачити, як авіаційна промисловість впроваджує в себе принципи сталості. Це може дозволити всім учасникам авіаційної промисловості взаємодіяти з пасажирами та підвищити обізнаність про те, як промисловість сприяє позитивному екологічному впливу.

Давайте заглибимося глибше, щоб зрозуміти, як блокчейн приносить користь авіаційній промисловості:

1) автоматизує процеси оплати: авіакомпанії, що використовують блокчейн, можуть надати безпечну платіжну систему, що дозволяє клієнтам здійснювати платежі більш впевнено. За допомогою блокчейну багато рутино-повторюваних процесів можна автоматизувати, таких як покупка

страхування подорожей, вирішення питань лояльності, сплата податків та зборів влади тощо. Навіть сам процес оплати буде надійнішим та ефективнішим, ніж коли-небудь;

2) покращує досвід клієнта: технологія блокчейн в авіакомпаніях може бути використана для поліпшення досвіду клієнта. Надавши пасажирам доступ до інформації про політ в реальному часі, токенізованих квитків, цифрового ведення записів, прозорості, цілісності даних та біометричної верифікації, не можна заперечувати той факт, що технологія блокчейн для аеропортів покращить рівень задоволення клієнтів. Це подальше спрощення процесів та мінімізація ризиків помилок;
3) зменшує залежність від посередників: близько 99% ринку непрямого продажу квитків контролюють туристичні агенти та інші посередники. Зрозуміло, що блокчейн для авіаційного ринку може допомогти вирішити це питання. Цей тренд у технологіях може допомогти скоротити витрати на посередників та збільшити доходи серед учасників мережі;
4) покращує терміни служби повітряних суден: за допомогою блокчейну авіакомпанії можуть відстежувати весь життєвий цикл літака, від процесу виробництва до процесу технічного обслуговування та ремонту, що дозволяє краще відстежувати та керувати частинами та компонентами повітряного судна;
5) оптимізує наземні операції: з впровадженням блокчейну авіаційна промисловість може зменшити витрати, оптимізувати процеси, покращити ефективність та збільшити безпеку. Блокчейн у авіаційній промисловості покращує відстежування, трасування та прозорість в операціях. Складність зменшується, і весь процес стає більш ефективним і спрощеним. Враховуючи все, блокчейн трансформує авіаційну промисловість, надаючи авіакомпаніям надійний, ефективний та економічний спосіб керування своєю діяльністю.

1.1.2. Алгоритми консенсусу

Вибір реалізації між різними типами блокчейнів безпосередньо пов'язаний з контекстом застосування. Залежно від цього вибору, буде надана перевага одній сім'ї алгоритмів консенсусу. Наприклад, використання алгоритмів, основаних на *Proof of Work (PoW)*, цікаве для публічних блокчейнів через їх велику масштабність; використання обчислювальної потужності як виборового ресурсу ускладнює компрометацію більше ніж 51% потужності мережі. Проте приватні мережі через свої обмежені ресурси є більш вразливими.

У блокчейн-мережах консенсус - це процес, за яким нові дані вважаються гідними бути доданими до попередніх записів; це відноситься до механізму валідації блоку. "Валідація", також відома як "перевірка", полягає в підпису блоку.

Ця згода на унікальний і загальний погляд на блокчейн повинна бути досягнута, навіть у випадку відсутності надійних вузлів, які також називають "византійськими" вузлами. Ці ненадійні вузли, як правило, мають довільну поведінку, включаючи зловмисні атаки (наприклад, атаки ботів та подвійне витрачання), помилки вузлів або також помилки з'єднання (які можуть призвести до розгалуження).

Протоколи консенсусу відрізняються, як зазначено вище, і можуть бути розподілені на дві великі сім'ї: вони або розробляються спільно, або конкурентно.

Доказ роботи (*PoW*). Це найпопулярніша схема, використовується у *Bitcoin*. Вузол отримує право додати новий блок до ланцюжка, якщо він може продемонструвати витрату певної кількості обчислювальної потужності. Для цього він повинен розв'язати обчислювально складну криптографічну головоломку. Саме цей алгоритм консенсусу був використаний в перших намаганнях розробити систему для купівлі авіаквитків на основі блокчейну, проте провівши дослідження прийшли до висновку, що використання цього алгоритму консенсусу не є доцільним.

Доказ власності (*PoS*). Реалізація в *Ethereum*, *BlackCoin* і *Peercoin*, доказ власності передбачає, що користувачі, які надіслали найбільшу кількість монет у

блокчейн, мають більше інтересів у забезпеченні його цілісності. У цьому випадку ймовірність вузла видобути наступний блок пов'язана з пропорцією його кількості монет до загальної надісланої в блокчейн кількості монет. Схема алгоритму продемонстрована на рис. 1.1.

Рис. 1.1. Схема алгоритму консенсусу *PoS*

Цей алгоритм був розроблений для вирішення основного і серйозного недоліку алгоритмів консенсусу, заснованих на *PoW*, а саме високого енергоспоживання для обчислення головоломки, що стало вже дуже великою проблемою в світі, оскільки шкодить екології. Після відкритя та дослідження цього алгоритму консенсусу, компанії намагались використовувати вже готові блокчейни для вирішення задачі, проте дійшли висновку що через загалальну популярність блокчейну, вони досить перегружені та вимагають комісії за транзакції, в деяких випадках чималу. Тому прийшли до висновку що краще розробляти власний блокчейн.

Оскільки розроблемий мною блокчейн використовується компанією він є приватним саме тому для його реалізації було використано алгоритм консенсусу *PoS*.

1.2. Приклади розробок

Рис. 1.2. Логотип компанії *Lufthansa*

Lufthansa (див. рис. 1.2) є однією з найбільших авіакомпаній у світі, і вона активно досліджує можливості використання блокчейн-технологій для різних цілей у туристичній галузі починаючи з 2022. Одним із напрямків досліджень компанії є використання блокчейн для купівлі авіаквитків. *Lufthansa* вважає, що блокчейн може допомогти покращити досвід покупців авіаквитків, зробивши процес покупки більш простим, швидким і безпечним.

У рамках своїх досліджень *Lufthansa* розробила прототип блокчейн-платформи для купівлі авіаквитків. Платформа використовує технологію блокчейн для зберігання інформації про бронювання авіаквитків. Це дозволяє пасажирам легко знайти та забронювати авіаквиток, а також отримати доступ до інформації про

своє бронювання в режимі реального часу. Компанія планує впровадити блокчейн-технології в свою діяльність у найближчі роки.

Lufthansa також досліджує можливість використання блокчейн для інших цілей у туристичній галузі, таких як:

1) бронювання готелів;

2) прокат автомобілів;

3) страхування подорожей;

4) обмін валют.

Компанія вважає, що блокчейн може допомогти покращити ефективність та прозорість туристичної галузі. Дослідження *Lufthansa* є одним із прикладів того, як блокчейн-технології можуть бути використані для революціонізації туристичної галузі.

Рис. 1.3. Логотип компанії *Winding Tree*

Winding Tree (див. рис. 1.3) є блокчейн-стартапом, спеціалізованим на туристичній галузі. Вони розробляють децентралізовану платформу для купівлі авіабілетів та інших туристичних послуг за допомогою криптовалют. Стартап був заснований у 2017 році групою підприємців, які прагнули створити більш справедливу та ефективну туристичну галузь. Вони вважають, що блокчейн може допомогти вирішити деякі з основних проблем галузі, такі як високі комісії, відсутність прозорості та централізована структура влади.

Платформа *Winding Tree* буде використовувати блокчейн для децентралізації туристичної галузі. Це означає, що туристи зможуть купувати авіаквитки, готельні номери та інші туристичні послуги безпосередньо у постачальників послуг, без необхідності платити комісію посередникам. Платформа також буде

використовувати блокчейн для забезпечення прозорості та відкритості інформації про ціни та умови.

Рис. 1.4. Логотип компанії *Air Asia*

Проект *Winding Tree* має партнерство з авіакомпанією *AirAsia* (див. рис. 1.4), яка використовує їхню технологію для продажу авіабілетів. Партнерство було оголошено в серпні 2023 року, а розпочнеться в 2024 році і буде поширюватися на всі рейси *AirAsia* по всьому світу. Партнерство *Winding Tree* з *AirAsia* є важливим кроком для компанії. *AirAsia* є однією з найбільших авіакомпаній у світі, і її партнерство з *Winding Tree* може допомогти компанії наблизитися до своєї мети з децентралізації туристичної галузі.

Під час партнерства *AirAsia* буде використовувати технологію *Winding Tree* для продажу авіаквитків за криптовалюту. Це дозволить пасажирам *AirAsia* купувати авіаквитки безпосередньо у компанії, без необхідності використовувати традиційні методи оплати, такі як кредитні картки або готівка.

Winding Tree все ще перебуває на ранніх стадіях розробки, але компанія вже залучила значні інвестиції від таких компаній, як *Binance Labs*, *NGC Ventures* та *1kx*. Компанія також має партнерства з рядом провідних туристичних компаній, таких як *Air France-KLM*, *AccorHotels* та *Booking.com*. Якщо компанія зможе реалізувати свої плани, вона може зробити подорожі більш доступними, справедливими та

ефективними для всіх. Ці партнерства є важливими для *Winding Tree*, оскільки вони допомагають компанії залучити більше постачальників послуг до своєї платформи. Чим більше постачальників послуг буде на платформі, тим більше туристів зможуть скористатися її перевагами.

Winding Tree є амбітним проектом, який має потенціал революціонізувати туристичну галузь. Якщо компанія зможе реалізувати свої плани, вона може зробити подорожі більш доступними, справедливими та ефективними для всіх.

Рис. 1.5. Логотип компанії *Singapore Airlines*

Singapore Airlines (*SIA*) – національна авіакомпанія Сінгапуру (див. рис. 1.5), створена 1 травня 1947 року. Кілька років поспіль отримує п’ять зірок з п’яти можливих від консалтингової компанії *Skytrax*.

Авіакомпанія має у своєму розпорядженні парк із понад 130 літаків, які здійснюють рейси до більш ніж 130 пунктів призначення у всьому світі. *SIA* є членом *Star Alliance*.

SIA також є лідером у галузі інновацій. Авіакомпанія першою впровадила ряд інноваційних технологій, включаючи:

1) першу у світі повністю автоматизовану систему реєстрації;
2) першу у світі систему обміну милями між авіакомпаніями;
3) першу у світі систему управління польотом на основі систем відстеження наземного руху (*ATM*).

SIA є однією з найуспішніших авіакомпаній у світі. Авіакомпанія має високий рівень прибутковості та є одним із найбільших роботодавців у Сінгапурі. Компанія є гордістю Сінгапуру та одним із найбільших її брендів. Авіакомпанія відіграє

важливу роль у розвитку економіки Сінгапуру та сприяє просуванню країни на міжнародній арені.

Singapore Airlines запустили платформу *KrisPay*, яка використовує технологію блокчейн для обміну миль на криптовалюту та зворотньо. Це дозволяє пасажирам використовувати майнджети для купівлі послуг та товарів. Платформа була запущена *Singapore Airlines* у 2023 році і є першою в світі блокчейн-платформою для авіакомпаній.

KrisPay дозволяє членам програми лояльності *Singapore Airlines*, обмінювати свої милі на криптовалюту. Це можна зробити за допомогою мобільного додатку *KrisPay*. Крім того, платформа *KrisPay* дозволяє використовувати криптовалюту для купівлі послуг та товарів. Також криптовалюту можна використовувати для оплати авіаквитків, готелів, їжі та інших витрат на подорож. Платформа має ряд переваг для пасажирів *Singapore Airlines*:

1) дозволяє пасажирам використовувати свої милі для купівлі широкого спектру послуг та товарів;
2) пропонує більш прозорі та ефективні умови обміну миль на криптовалюту;
3) використовує технологію блокчейн від *Microsoft Azure* для забезпечення безпеки та надійності операцій.

Платформа *KrisPay* є важливим кроком для *Singapore Airlines*. Вона дозволяє компанії запропонувати своїм клієнтам нові можливості використання своїх миль і сприяє популяризації криптовалют.

Рис. 1.6. Логотип компанії *Universal Air Travel Plan*

UATP (*Universal Air Travel Plan*) – це глобальна платіжна мережа (див. рис. 1.6), яка спеціалізується на оплаті авіаквитків та інших туристичних витрат. Вона була заснована в 1936 році і є однією з найстаріших платіжних мереж у світі.

У мережі *UATP* беруть участь понад 260 авіакомпаній, які обслуговують понад 140 країн світу. *UATP* також має партнерські відносини з багатьма іншими компаніями, що працюють у туристичній галузі, такими як готелі, туроператори та страхові компанії. Мережа пропонує широкий спектр послуг для своїх учасників, включаючи:

1) оплату авіаквитків та інших туристичних витрат;
2) обмін миль;
3) оплату податків і зборів;
4) страхування подорожей.

UATP є одним із найбільших платіжних операторів у світі. У 2022 році через мережу було перераховано понад 20 мільярдів доларів США. Ось деякі з ключових переваг використання:

1) універсальність: *UATP* є глобальною мережею, яка підтримується багатьма авіакомпаніями та іншими компаніями, що працюють у туристичній галузі;
2) безпека: *UATP* використовує передові технології безпеки для захисту даних своїх учасників;
3) комфорт: *UATP* пропонує прості та зручні способи оплати авіаквитків та інших туристичних витрат.

UATP є важливою частиною туристичної галузі. Мережа допомагає людям подорожувати по всьому світу, надаючи їм доступ до широкого спектру послуг та переваг.

UATP (*Universal Air Travel Plan*) випустила блокчейн-платформу "*UATP Chain*". Ця платформа спрямована на полегшення та упрощення платежів за авіаквитки для компаній та пасажирів.

UATP Chain - це блокчейн-платформа, розроблена *UATP*, спеціалізованою платіжною компанією для авіаційної промисловості. Платформа була запущена в 2023 році і спрямована на полегшення та упрощення платежів за авіаквитки для компаній та пасажирів.

UATP Chain використовує технологію блокчейн від *IBM Hyperledger Fabric* для забезпечення безпеки та надійності платежів. Платформа також може допомогти в скороченні витрат на платежі та покращенні прозорості операцій.

UATP Chain пропонує ряд переваг для компаній та пасажирів:

1) скоротити витрати на платежі;
2) покращити прозорість операцій;
3) здійснювати платежі за авіаквитки швидше та простіше.

UATP Chain є важливим кроком для *UATP*. Вона дозволяє компанії запропонувати своїм клієнтам нові можливості для здійснення платежів за авіаквитки і сприяє популяризації технології блокчейн в авіаційной промисловості.

Платформа підтримує низку криптовалют, включаючи *Bitcoin, Ethereum* та *Tether*. *UATP Chain* має намір розширити свої можливості в майбутньому, щоб включати інші види платежів, пов'язаних з авіаперевезеннями, такі як бронювання готелів та прокат автомобілів.

1.3. Висновки до розділу

У цьому розділі було проведенно глибокий аналіз можливостей використання технології блокчейн для оптимізації процесу придбання авіаквитків. Важливо відзначити ключові характеристики блокчейну, такі як децентралізація, незмінність та підвищена безпека, що робить його ідеальним інструментом для впровадження у сферу авіаційної промисловості.

Основні можливості використання технології блокчей в авіаційній промисловості включають:

1) об'єднання різних сегментів галузі для створення безшовного досвіду подорожування для клієнтів;
2) покращення задоволення клієнтів та зменшення витрат на неефективність.

Загалом, використання технології блокчейн у сфері купівлі авіаквитків обіцяє значні поліпшення у безпеці, ефективності та цифровому досвіді подорожування для всіх учасників авіаційної промисловості.

Також було розглянуто важливість вибору алгоритму консенсусу при впровадженні блокчейну в авіаційну промисловість. Для приватних блокчейнів, запропонованих у цій роботі, алгоритм *Proof of Stake* (*PoS*) є більш ефективним.

Враховуючи, що консенсус у блокчейні визначає, як нові дані додаються до попередніх записів, вибір правильного алгоритму є ключовим етапом у розгортанні блокчейну в авіаційній промисловості. Цей вибір повинен бути узгоджений зі специфічними потребами та контекстом застосування.

Також, надано приклади розробок у сфері авіаційної промисловості, де використання блокчейн-технологій може призвести до значних поліпшень. Компанія *Lufthansa*, що є одним з найбільших гравців у цій галузі, активно досліджує можливості впровадження блокчейну в свою діяльність. Окрім того є ще декілька стартапів та компаній які також розробляють свої інструменти на основі блокчейну для авіаційної галузі, такі як - *Winding Tree*, *AirAsia*, *Singapore Airlines*, *Universal Air Travel Plan*. Однією з ключових областей досліджень та розробок є використання блокчейну для купівлі авіабілетів.

РОЗДІЛ 2
ВИКОРИСТАННЯ ПРОГРАМНИХ ЗАСОБІВ ТА БІБЛІОТЕК

2.1. Середовище розробки *Visual studio code*

Для написання програмної системи для купівлі авіаквитків на основі блокчейну було обрано *Visual Studio Code*, оскільки:

1) це потужний і гнучкий редактор коду, який надає широкий спектр функцій, необхідних для написання складних програм. До цих функцій належать автозаповнення коду, перевірка помилок, розбивка коду на блоки та багато іншого. Це може допомогти вам написати чистий і добре структурований код, який легше читати та підтримувати.
2) підтримує широкий спектр мов програмування, зокрема *Python, HTML, CSS, JS*. Всі ці мови були використані під час написання кваліфікаційної роботи, використовуючи один і той же редактор коду для написання блокчейну та веб-інтерфейсу.
3) має велике та активне співтовариство користувачів. Це означає, що ви завжди можете отримати допомогу, якщо у вас виникнуть проблеми з написанням коду.
4) є безкоштовним і відкритим програмним забезпеченням. Це означає, що ви можете використовувати його безкоштовно і внести свій вклад у його розвиток.

2.2. Мова програмування *Python*

Для написання коду я обрав мову програмування *Python*, оскільки це чудовий вибір з кількох причин:

1) це мова загального призначення, яка використовується для широкого спектру завдань, включаючи написання блокчейну. Це означає, що

розробники, які вже знають *Python*, можуть використовувати свої навички для написання блокчейну, не навчаючись нової мови;

2) це мова з високим рівнем абстракції, яка робить її легкою для читання та розуміння. Це важливо для написання блокчейну, де код повинен бути простим і зрозумілим, щоб його можна було легко перевірити на наявність помилок;
3) це мова з високою продуктивністю, яка може використовуватися для написання швидкодіючих блокчейн-додатків;
4) це мова з відкритим кодом, яка підтримується великою спільнотою розробників. Це означає, що існує велика кількість ресурсів, доступних для розробників *Python*, включаючи книги, статті та форуми підтримки;
5) використовується в багатьох галузях, включаючи веб-розробку, наукові дослідження, обробку даних, штучний інтелект, машинне навчання та інше. Це дозволяє вам вибрати те напрямок, який вас цікавить, і впевнено працювати над ним;
6) використовується багатьма великими компаніями та організаціями, такими як *Google, Facebook, Instagram, Spotify* та інші. Він також є основною мовою для багатьох стартапів;
7) навички роботи з *Python* корисні для подальшого розвитку у сфері програмування, аналізу даних, штучного інтелекту та багатьох інших сферах;
8) добре поєднується з іншими мовами програмування, що може бути важливо для роботи з великими проектами або в областях, де використовуються різні технології.

2.3. Бібліотеки та фреймворки

2.3.1. Бібліотека *Cryptodome*

З бібліотеки *Cryptodome* використовується алгоритм хешування *SHA*256, який використовується для обчислення хеш-суми для даних. Використання хеш-функцій, таких як *SHA*256, є важливим для безпеки даних. Ці функції приймають вхідні дані (наприклад, повідомлення або файли) та обчислюють унікальний хеш-код для них. Навіть найменша зміна вхідних даних призводить до істотної зміни у вихідному хеші. Криптографічні функції, які використовуються для роботи з *RSA*-шифруванням та цифровим підписом моїй програмній системі:

а) модуль *PublicKey*: він надає можливість працювати з криптографічними ключами. У моєму випадку, використовується для генерації та роботи з *RSA*-ключами;

б) модуль *Cryptodome.Signature*: він містить класи та функції для генерації та перевірки цифрових підписів. У моєму випадку, використовується алгоритм *PKCS1_v1_5* для цифрового підпису з *RSA*-ключами.

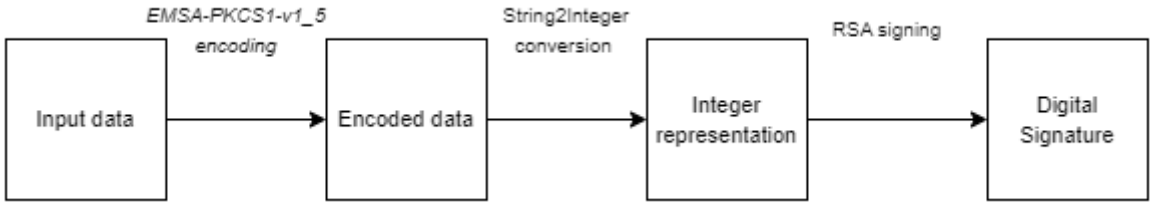

Рис. 2.1. Процес генерації підпису *PKCS1_v1_5*

Алгоритм *PKCS1_v1_5* використовує кодувальну схему *EMSA-PKCS1-v1_5* для створення цифрового підпису. На рис. 2.1 продемонстровано процес генерації підпису *PKCS1_v1_5*. Загалом він приймає вхідні дані та виконує підписування в трьох кроках. На першому кроці вхідні дані кодуються за допомогою методу кодування *EMSA-PKCS1-v1_5*. Другий крок - перетворення закодованого рядка в ціле число. Останній крок - підписання цілочисельного представлення.

Підписування включає шифрування з використанням приватного ключа алгоритму *RSA* для генерації цифрового підпису.

2.3.2. Бібліотеки *Json* та *Jsonpickle*

JSON (JavaScript Object Notation) - це легкий формат обміну даними, який зручний для читання та запису людьми, а також легко оброблюється комп'ютерними системами. У *Python* є вбудований модуль *json*, який надає інтерфейс для роботи з даними у форматі *JSON*. Основні операції, які виконуються в моїх роботі з модулем *json*:

1) серіалізація (кодування): перетворення об'єктів *Python* (наприклад, словників, списків) у формат *JSON*. Це може бути корисним для збереження даних або їх передачі через мережу;
2) десеріалізація (декодування): перетворення *JSON*-строки у відповідний об'єкт *Python*;
3) робота з файлами: модуль *json* також надає методи для читання та запису *JSON*-даних у файли.

Ці операції дозволяють легко обробляти та обмінюватися даними у форматі *JSON* в програмах на *Python*. *JSON* використовується широко, особливо у веб-розробці та обробці даних.

jsonpickle - це бібліотека *Python*, яка надає можливість серіалізувати та десеріалізувати складні об'єкти *Python* в формат *JSON*. Однак, на відміну від вбудованого модулю *json*, він дозволяє обробляти більш широкий спектр об'єктів, включаючи класи, екземпляри класів, та багато інших типів даних. Основні можливості *jsonpickle*:

1) підтримка складних об'єктів: *jsonpickle* дозволяє серіалізувати об'єкти, що містять вкладені структури даних, такі як списки, словники, класи тощо;
2) підтримка користувацьких класів: Вона дозволяє серіалізувати та десеріалізувати екземпляри класів, включаючи їх атрибути та методи;

3) підтримка специфічних для *Python* об'єктів: *jsonpickle* може обробляти специфічні для *Python* об'єкти, такі як модулі, виключення, типи, функції тощо;
4) можливість керувати серіалізацією: *jsonpickle* надає можливість впливати на процес серіалізації шляхом надання спеціальних методів для класів;
5) підтримка власних конвертерів: *jsonpickle* дозволяє вам визначити свої конвертери для специфічних типів даних або класів.

2.3.3. Бібліотека *Requests*

Requests - це популярна бібліотека *Python*, яка надає простий та зручний інтерфейс для взаємодії з *HTTP*-запитами. З її допомогою ви можете відправляти запити до вебсерверів і отримувати відповіді. Типовий потік запитів зображено на рис. 2.2. Основні можливості *Requests*:

1) відправлення *HTTP*-запитів: надає можливість відправляти *GET*, *POST*, *PUT*, *DELETE* та інші типи запитів до серверів;
2) робота з параметрами та заголовками: надає можливість передавати параметри та заголовки у своїх запитах;
3) отримання відповіді від сервера: після відправлення запиту, повертає об'єкт *Response*, з яким можна працювати;
4) робота з кукісами та сесіями: *requests* дозволяє керувати кукісами та створювати сесії для взаємодії з вебсайтами, які вимагають аутентифікації;
5) обробка редиректів та обривів з'єднання: *requests* автоматично обробляє редиректи та може керувати таймаутами.

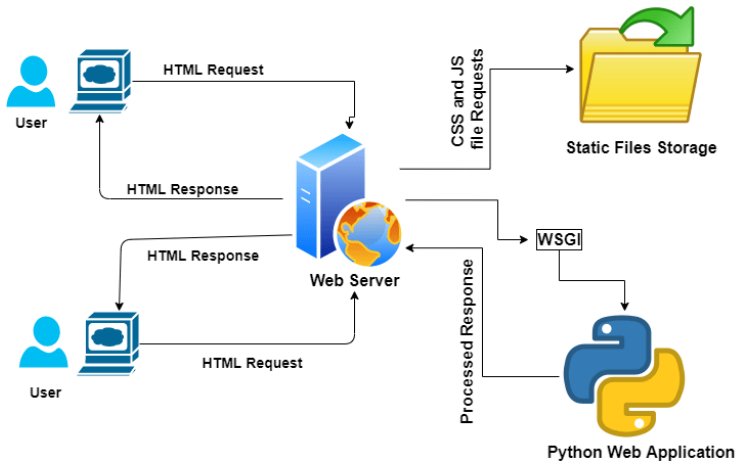

Рис. 2.2. Типовий потік запитів

2.3.4. Бібліотека *Copy*

Модуль *copy* в *Python* надає можливість створювати копії об'єктів. Його основна функція - це надання можливості створювати глибокі копії об'єктів, що означає, що копії не посилаються на ті ж самі об'єкти в пам'яті, як і оригінал. Основні функції та методи модуля copy включають:

а) *copy*(): ця функція створює поверхневу копію об'єкта, що означає, що вона копіює об'єкт, але не копіює об'єкти, на які посилається;

б) *deepcopy*(): ця функція створює глибоку копію об'єкта, що означає, що вона копіює об'єкт та всі об'єкти, на які він посилається, та всі об'єкти, на які вони посилаються, рекурсивно;

в) *copy.copy*() та *copy.deepcopy*() для користувацьких класів: ці методи можна використовувати для копіювання екземплярів користувацьких класів, викликаючи методи *__copy__* та *__deepcopy__* відповідно.

Відмінність *copy* (*Shallow copy*) від *Deep copy* зображено на рис. 2.3.

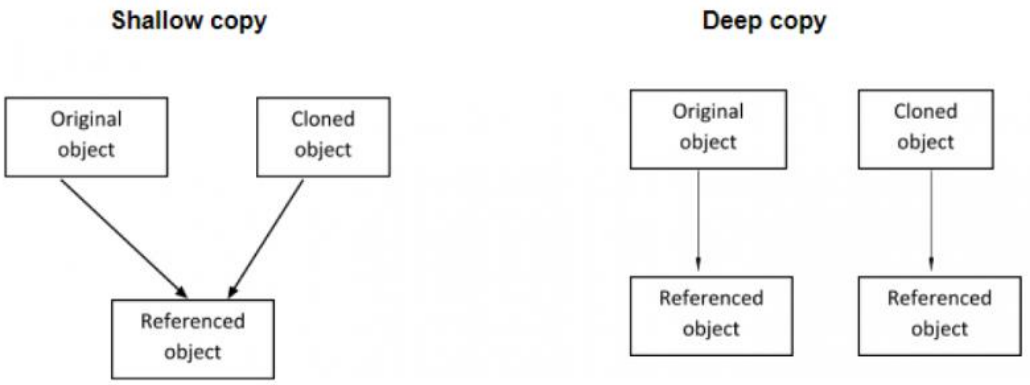

Рис. 2.3. Відмінність *Shallow copy* від *Deep copy*

2.3.5. Фрейморк *Flask* та *Flask_classful*

В моєму випадку використовуються дві бібліотеки для розробки вебдодатків на мові *Python*:

1) фреймворк *Flask* - це легковагий вебфреймворк для *Python*, який дозволяє швидко створювати вебдодатки. У цьому коді *Flask* використовується для створення основної програми додатку;
2) доповнення *Flask-classful*: надає можливість використовувати класи для організації рут та переглядів у вебдодатку.

Короткий опис функцій та об'єктів у моїй системі:

1) клас *FlaskView* є основою для визначення переглядів у додатку. Він надає методи, такі як *route*, які дозволяють визначити маршрути та їх поведінку;
2) декоратор *route* використовується для визначення маршруту та пов'язаної з ним функції-обробника. У контексті *FlaskView* це означає, що функція буде викликатися, коли буде отримано відповідний запит;
3) об'єкт *Flask* представляє основний додаток *Flask*. Він використовується для запуску та конфігурації вебдодатку;
4) функція *jsonify* з бібліотеки *flask* використовується для повернення *JSON*-відповіді;
5) об'єкт *request* надає інформацію про поточний *HTTP*-запит, таку як метод, аргументи, заголовки тощо;

6) функція *render_template* використовується для рендерінгу *HTML*-шаблонів від *Flask*-додатку.

На рис. 2.4 зображено використання *Flask* для взаємодії з блокчейном в моїй системі.

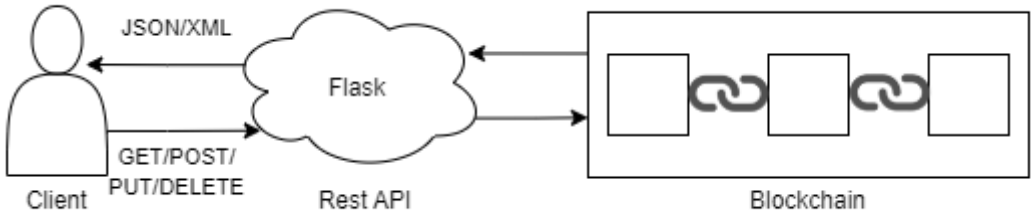

Рис. 2.4. Використання *Flask* для взаємодії з блокчейном

2.3.6. Бібліотека *Threading*

threading - це вбудований модуль в *Python*, який надає можливість створювати та керувати потоками в програмі. Потоки дозволяють виконувати різні завдання паралельно, що може покращити продуктивність додатку. Основні можливості *threading*:

1) створення потоків: модуль надає клас *Thread*, який дозволяє створити новий потік в програмі;
2) синхронізація потоків: модуль також надає засоби для синхронізації роботи потоків, щоб уникнути конфліктів та проблем з доступом до ресурсів;
3) робота зі таймаутами та інтервалами: *threading* дозволяє встановлювати таймаути та інтервали для контролю часу виконання потоків;
4) пул потоків: модуль дозволяє створювати пул потоків для ефективного використання ресурсів.

2.3.7. Бібліотека *P2pnetwork*

Бібліотека *p2pnetwork* використовується для створення вузла (*Node*) в мережі *peer-to-peer* (*P2P*) у середовищі *Python*.

Основні компоненти та функції які використовуються в моєму прикладі включають:

1) бібліотека *p2pnetwork* для реалізації мереж *P2P* у *Python*;
2) клас *Node* є частиною бібліотеки *p2pnetwork* і використовується для створення та керування вузлами (комп'ютерами або обчислювальними пристроями), які можуть спілкуватися один з одним в мережі *P2P*. Клас *Node* надає методи та функціональність для побудови і підтримки таких мереж.

2.3.8. Бібліотека *Uuid*

uuid - це модуль в стандартній бібліотеці *Python*, який надає функції для генерації та роботи з унікальними ідентифікаторами. *UUID* (унікальний ідентифікатор універсального вигляду) - це 128-бітне число, яке гарантує унікальність.

Деякі основні функції та методи модуля *uuid*:

1) *uuid1*(): генерує *UUID* на основі часу та *MAC*-адреси. Гарантує унікальність, але не абсолютну випадковість (див. рис. 2.5);
2) *uuid4*(): генерує випадковий *UUID*. Не гарантує унікальність, але в ймовірнісному сенсі майже завжди єдиний;
3) конструктор *UUID*(), який дозволяє створити *UUID* зі строки у специфічному форматі;
4) *hex*: атрибут, який містить 32-символьний шістнадцятковий рядок, що представляє *UUID*;
5) *urn*: атрибут, який містить *UUID* у форматі *URN* (*Uniform Resource Name*).

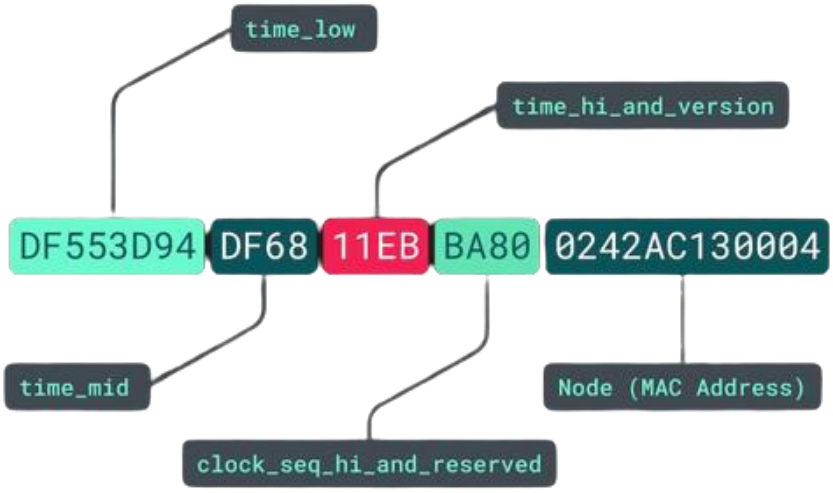

Рис. 2.5. Елементи сгенерованого *UUID*

2.4. Висновки до розділу

У цьому розділі диплому розглянуто важливі бібліотеки та інструменти, що використовуються для розробки програмної системи для купівлі авіаквитків на основі блокчейну на мові *Python* з використанням середовища *Visual Studio Code*.

Visual Studio Code надає широкий спектр функцій, що полегшують написання чистого та структурованого коду. Це ефективний інструмент для розробки програмного забезпечення, зокрема для проектів, пов'язаних із блокчейном та криптографією.

Python є потужною та гнучкою мовою програмування, яка чудово підходить для різних областей. В даному випадку, вона використовується для реалізації блокчейну та забезпечення його безпеки через публічні та приватні ключі.

Бібліотека *Cryptodome* використовується для роботи з хешуванням *SHA*256, а також для роботи з *RSA*-ключами та цифровим підписом, що забезпечує безпеку обміну даними.

Бібліотеки *Json* та *Jsonpickle* використовуються для роботи з форматом *JSON*, що є універсальним та зручним для обміну даними. Вони надають засоби для

серіалізації та десеріалізації об'єктів *Python*, включаючи складні структури та користувацькі класи.

Бібліотека *Copy* дозволяє створювати копії об'єктів, включаючи глибокі копії, що дозволяє зберігати незалежні копії об'єктів.

Flask використовується для створення вебдодатків, визначаючи маршрути, обробники запитів та відображення.

Бібліотека *request* є важливою для взаємодії з *API* та вебскрапінгу, що робить її корисною для автоматизації тестування вебдодатків та інших сценаріїв, де потрібно взаємодіяти з вебресурсами.

Клас *Node* утворює основу для розробки *P2P*-додатка у *Python*, де можна використовувати бібліотеку *p2pnetwork* для створення вузлів та реалізації комунікації у мережі *P2P*.

Бібліотека *uuid* використовується для генерації та роботи з унікальними ідентифікаторами (*UUID*). Ці ідентифікатори гарантують унікальність та можуть бути використані для ідентифікації різних об'єктів у системі. Вона надає різні методи для створення *UUID*, зокрема на основі часу та *MAC*-адреси (*uuid1*) або випадковим чином (*uuid4*). Ця бібліотека допомагає забезпечити, що кожен об'єкт чи елемент у системі матиме унікальний ідентифікатор, що ускладнює конфлікти та дублювання.

РОЗДІЛ 3
ОПИС РЕАЛІЗАЦІЇ ЗАВДАННЯ

Розроблена мною програмна система розділена на класи, кожен з яких виконує важливу функціональну роль у системі (див. рис. 3.1).

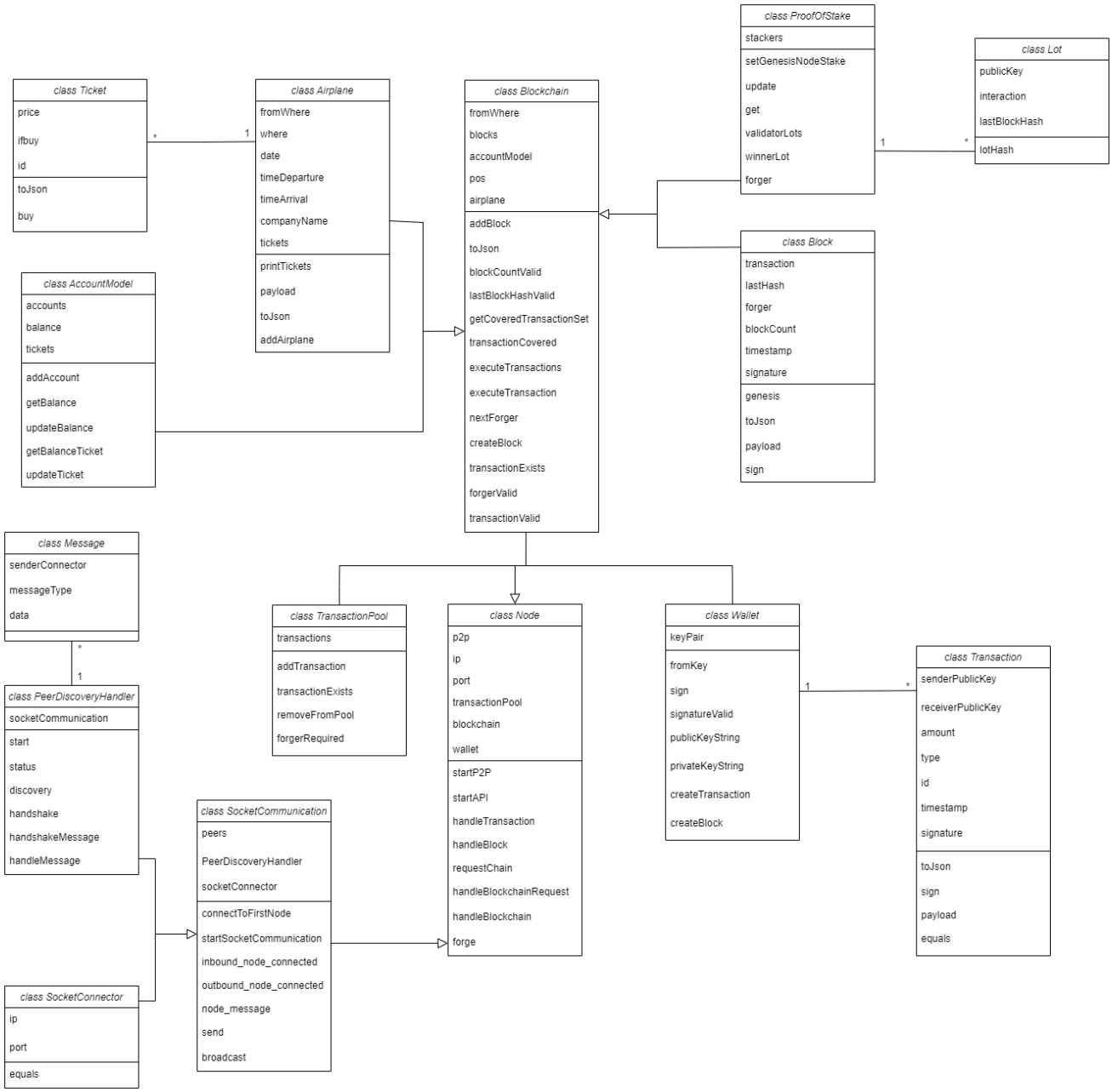

Рис. 3.1. Класи програмної системи

3.1. Розробка програмної системи на основі блокчейну

Файл *Transaction.py*

```
class Transaction():
    def __init__(self,senderPublicKey,receiverPublicKey,amount,type):
        self.senderPublicKey = senderPublicKey
        self.receiverPublicKey = receiverPublicKey
        self.amount = amount
        self.type = type
        self.id = uuid.uuid1().hex #16
        self.timestamp = time.time()
        self.signature = ''
    def toJson(self):
        return self.__dict__
    def sign(self,signature):
        self.signature = signature
    def payload(self):
        jsonRepresentation = copy.deepcopy(self.toJson())
        jsonRepresentation['signature'] = ''
        return jsonRepresentation
    def equals(self,transaction):
        if self.id == transaction.id:
            return True
        else:
            return False
```

Код містить опис класу *Transaction*, який представляє транзакцію в блокчейні:

1) конструктор *__init__*(*self, senderPublicKey, receiverPublicKey, amount, type*): він ініціалізує атрибути об'єкта *Transaction*;
2) метод *toJson*(*self*): повертає словник, який представляє об'єкт *Transaction* у вигляді *JSON*;

3) метод *sign*(*self, signature*): встановлює підпис транзакції. Приймає параметр *signature*, який є підписом транзакції;
4) метод *payload*(*self*): повертає копію *JSON*-представлення транзакції, але без підпису. Ця функція видаляє підпис перед подальшою обробкою транзакції;
5) метод *equals*(*self, transaction*): порівнює ідентифікатори поточної транзакції з іншою транзакцією, переданою як аргумент. Якщо ідентифікатори співпадають, повертає *True*, інакше - *False*. Це використовується, ймовірно, для порівняння транзакцій і перевірки їх унікальності.

Файл *Wallet.py*

```
class Wallet():
    node = None
    def __init__(self):
        self.keyPair = RSA.generate(2048)
    def __getstate__(self):
        return self.__dict__
    def __reduce__(self):
        return (self.__class__, ())
    def __setstate__(self, state):
        self.__dict__ = state
    def fromKey(self, file):
        key = ''
        with open(file, 'r') as keyfile:
            key = RSA.import_key(keyfile.read())
        self.keyPair = key
    def sign(self, data):
        dataHash = BlockchainUtils.hash(data)
        signatureSchemeObject = PKCS1_v1_5.new(self.keyPair)
        signature = signatureSchemeObject.sign(dataHash)
        return signature.hex()
```

```
        @staticmethod
        def signatureValid(data,signature,publicKeyString):
            signature = bytes.fromhex(signature)
            dataHash = BlockchainUtils.hash(data)
            publicKey = RSA.import_key(publicKeyString)
            signatureSchemeObject = PKCS1_v1_5.new(publicKey)
            signatureValid = signatureSchemeObject.verify(dataHash,signature)
            return signatureValid
        def publicKeyString(self):
            publicKeyString =
self.keyPair.publickey().exportKey('PEM').decode('utf-8')
            return publicKeyString
        def privateKeyString(self):
            privateKeyString = self.keyPair.exportKey('PEM').decode('utf-8')
            return privateKeyString
        def createTransaction(self,receiver,amount, type):
            transaction = Transaction(self.publicKeyString(),receiver,amount,type)
            signature = self.sign(transaction.payload())
            transaction.sign(signature)
            return transaction
        def createBlock(self, transaction, lashHash, blockCount):
            block = Block(transaction, lashHash, self.publicKeyString(),
blockCount)
            signature = self.sign(block.payload())
            block.sign(signature)
            return block
```

Клас *Wallet* дозволяє створювати гаманці, генерувати ключі, підписувати дані, створювати транзакції та блоки. Також він надає можливість взаємодіяти з ключами через файли і використовувати їх для підпису даних:

1) конструктор *__init__*(*self*): в ньому генерується нова пара ключів *RSA* (*keyPair*) з довжиною 2048 бітів;
2) метод *fromKey*(*self, file*): зчитує ключі з файлу *file* і встановлює їх у *keyPair*. Приймає шлях до файлу з ключами;
3) метод *sign*(*self, data*): підписує *data* за допомогою приватного ключа гаманця. Спочатку обчислює хеш *data*, а потім підписує його з використанням приватного ключа. Повертає підпис у вигляді шістнадцяткового рядка;
4) метод *signatureValid*(*data, signature, publicKeyString*): перевіряє, чи є підпис *signature* дійсним для *data* з використанням публічного ключа *publicKeyString*;
5) метод *publicKeyString*(*self*): повертає публічний ключ гаманця у форматі *PEM* у вигляді рядка;
6) метод *privateKeyString*(*self*): повертає приватний ключ гаманця у форматі *PEM* у вигляді рядка;
7) метод *createTransaction*(*self, receiver, amount, type*): створює нову транзакцію за допомогою публічного ключа гаманця як відправника. Підписує транзакцію та повертає її;
8) метод *createBlock*(*self, transaction, lastHash, blockCount*): створює новий блок з вказаною транзакцією, останнім хешем блоку та номером блоку. Підписує блок та повертає його.

Файл *TransactionPool.py*

```
class TransactionPool():
  def __init__(self):
    self.transactions = []
  def addTransaction(self,transaction):
    self.transactions.append(transaction)
  def transactionExists(self, transaction):
    for poolTransaction in self.transactions:
```

```
            if poolTransaction.equals(transaction):
                return True
        return False
    def removeFromPool(self, transactions):
        newPoolTransactions = []
        for poolTransaction in self.transactions:
            insert = True
            for transaction in transactions:
                if poolTransaction.equals(transaction):
                    insert = False
            if insert == True:
                newPoolTransactions.append(poolTransaction)
        self.transactions = newPoolTransactions
    def forgerRequired(self):
        if len(self.transactions)>=3: #block for each tx
            return True
        else:
            return False
```

Цей код визначає клас *TransactionPool*, який представляє собою пул транзакцій у блокчейні. Ось що роблять методи цього класу:

1) конструктор *__init__*(*self*): ініціалізує атрибут *transactions* як порожній список, який буде використовуватися для зберігання транзакцій;
2) метод *addTransaction*(*self, transaction*): додає транзакцію *transaction* до пулу транзакцій;
3) метод *transactionExists*(*self, transaction*): перевіряє, чи існує вже така транзакція у пулі. Проходиться по усіх транзакціях у пулі і порівнює їх з переданою транзакцією *transaction*. Якщо знаходить транзакцію, яка має такий самий ідентифікатор (*equals*(*transaction*) повертає *True*), то повертає *True*, інакше – *False*;

4) метод *removeFromPool*(*self, transactions*): видаляє вказані транзакції з пулу транзакцій. Приймає список транзакцій *transactions* та оновлює список транзакцій *self.transactions*, виключаючи ті, які зустрічаються у переданому списку;
5) метод *forgerRequired*(*self*): перевіряє, чи потрібно створити новий блок на основі транзакцій у пулі. У цьому випадку, якщо кількість транзакцій у пулі більше або дорівнює 3, то повертає *True*, в іншому випадку - *False*.

Файл *Block.py*

```
class Block():
    def __init__(self, transactions, lastHash, forger, blockCount):
      self.transactions = transactions
      self.lastHash = lastHash
      self.forger = forger
      self.blockCount = blockCount
      self.timestamp = time.time()
      self.signature = ''
    @staticmethod
    def genesis():
       genesisBlock = Block([],'genesisHash','genesis',0)
       genesisBlock.timestamp = 0
       return genesisBlock
    def toJson(self):
        data = {}
        data['lashHash'] = self.lastHash
        data['forget'] = self.forger
        data['blockCount'] = self.blockCount
        data['timestamp'] = self.timestamp
        data['signature'] = self.signature
        jsonTransaction = []
```

```
        for transaction in self.transactions:
            jsonTransaction.append(transaction.toJson())
        data['transactions'] = jsonTransaction
        return data
    def payload(self):
        jsonRepresentation = copy.deepcopy(self.toJson())
        jsonRepresentation['signature'] = ''
        return jsonRepresentation
    def sign (self, signature):
        self.signature = signature
```

Цей файл включає в себе опис класу *Block*, який представляє блок у блокчейні або подібній системі. Ось що роблять методи та функції:

1) конструктор *__init__*(*self, transactions, lastHash, forger, blockCount*): він ініціалізує атрибути об'єкта *Block* відповідно до переданих параметрів:
   - *transactions* (список транзакцій);
   - *lastHash* (хеш попереднього блоку);
   - *forger* (ідентифікатор генератора блоку);
   - *blockCount* (номер блоку);
   - також встановлює час створення блоку (*timestamp*);
   - та підпис блоку (*signature*);
2) метод *genesis*(): Статичний метод, який створює та повертає "генезис" блок (перший блок у блокчейні) з пустим списком транзакцій, фіксованим значенням хешу та генератора, а також з номером блоку рівним 0;
3) метод *toJson*(*self*): Перетворює об'єкт *Block* у словник, що представляє його у форматі *JSON*;
4) метод *payload*(*self*): Створює копію *JSON*-представлення блоку, але з очищеним полем *signature*. Це використовується для подальшої обробки блоку;
5) метод *sign*(*self, signature*): Встановлює підпис блоку. Приймає параметр *signature*, який є підписом блоку.

Файл *Blockchain.py*

```
class Blockchain():
    def __init__(self):
        self.blocks = [Block.genesis()]
        self.accountModel = AccountModel()
        self.pos = ProofOfStake()
        self.airplane = Airplane()
    def addBlock(self,block):
        self.executeTransactions(block.transactions)
        self.blocks.append(block)
    def toJson(self):
        data={}
        jsonBlocks = []
        for block in self.blocks:
            jsonBlocks.append(block.toJson())
        data['blocks'] = jsonBlocks
        return data
    def blockCountValid(self,block):
        if self.blocks[-1].blockCount == block.blockCount - 1:
            return True
        else:
            return False
    def lashBlockHashValid(self,block):
        latestBlockchainBlockHash = BlockchainUtils.hash(self.blocks[-
1].payload()).hexdigest()
        if latestBlockchainBlockHash == block.lastHash:
            return True
        else:
            return False
```

```
        def getCoveredTransactionSet(self, transactions):
            coveredTransactions = []
            for transaction in transactions:
                if self.transactionCovered(transaction):
                    coveredTransactions.append(transaction)
                else:
                    print('Transaction is not covered by sender')
            return coveredTransactions
        def transactionCovered(self, transaction):
            if transaction.type == 'EXCHANGE':
                return True
            senderBalance
=  self.accountModel.getBalance(transaction.senderPublicKey)
            if senderBalance >= transaction.amount:
                return True
            else :
                return False
        def executeTransactions(self, transactions):
            for transaction in transactions:
                self.executeTransaction(transaction)
        def executeTransaction(self, transaction):
            if transaction.type == 'BUYTICKET':
                sender = transaction.senderPublicKey
                receiver = transaction.receiverPublicKey
                amount = self.airplane.tickets[0].price
                self.accountModel.updateBalace(sender, -amount)
                self.accountModel.updateBalace(receiver, amount)
                self.accountModel.updateTicket(sender,self.airplane.tickets[0].id)
                self.airplane.tickets[0].buy()
            elif transaction.type == 'STAKE':
```

```
            receiver = transaction.receiverPublicKey
            if sender == receiver:
                amount = transaction.amount
                self.pos.update(sender, amount)
                self.accountModel.updateBalace(sender, -amount)
        else:
            sender = transaction.senderPublicKey
            receiver = transaction.receiverPublicKey
            amount = transaction.amount
            self.accountModel.updateBalace(sender, -amount)
            self.accountModel.updateBalace(receiver, amount)
    def nextForger(self):
        lastBlockHash = BlockchainUtils.hash(self.blocks[-
1].payload()).hexdigest()
        nextForger = self.pos.forger(lastBlockHash)
        return nextForger
    def createBlock(self, transactionFromPool, forgerWallet):
        coveredTransactions =
self.getCoveredTransactionSet(transactionFromPool)
        self.executeTransactions(coveredTransactions)
        newBlock = forgerWallet.createBlock(coveredTransactions,
                                    BlockchainUtils.hash(self.blocks[-
1].payload()).hexdigest(), len(self.blocks))
        self.blocks.append(newBlock)
        return newBlock
    def transactionExists(self, transaction):
        for block in self.blocks:
            for blockTransaction in block.transactions:
                if transaction.equals(blockTransaction):
                    return True
```

```
        forgerPublicKey = self.pos.forger(block.lastHash)
        proposedBlockForger = block.forger
        if forgerPublicKey == proposedBlockForger:
            return True
        else:
            return False
    def transactionValid(self, transactions):
        coveredTransactions = self.getCoveredTransactionSet(transactions)
        if len(coveredTransactions) == len(transactions):
            return True
        return False
```

Цей файл містить клас *Blockchain*, який реалізує основну логіку блокчейну. Ось що роблять методи та функції цього класу:

1) конструктор *__init__*(*self*): він ініціалізує атрибути об'єкта *Blockchain*, включаючи перший блок (генезис), об'єкт моделі облікового запису *AccountModel*, об'єкт *ProofOfStake* та об'єкт *Airplane*;
2) метод *addBlock*(*self, block*): додає новий блок до ланцюга. Викликає метод *executeTransactions* для виконання транзакцій перед додаванням блоку;
3) метод *toJson*(*self*): перетворює об'єкт *Blockchain* у словник, що представляє його у форматі *JSON*;
4) метод *blockCountValid*(*self, block*): перевіряє, чи правильно встановлений номер блоку у вхідному блоку *block* в порівнянні з останнім блоком у ланцюгу;
5) метод *lashBlockHashValid*(*self, block*): перевіряє, чи правильно встановлений хеш останнього блоку у вхідному блоку *block* в порівнянні з останнім блоком у ланцюгу;
6) метод *getCoveredTransactionSet*(*self, transactions*): повертає список транзакцій, які можна виконати на основі поточного стану рахунків;
7) метод *transactionCovered*(*self, transaction*): перевіряє, чи може транзакція бути виконана, враховуючи баланс відправника;

8) метод *executeTransactions*(*self, transactions*): виконує всі транзакції в переданому списку;
9) метод *executeTransaction*(*self, transaction*): виконує одну транзакцію, залежно від її типу;
10) метод *nextForger*(*self*): визначає наступного генератора блоку за допомогою *Proof of Stake*;
11) метод *createBlock*(*self, transactionFromPool, forgerWallet*): створює новий блок з транзакціями з пулу транзакцій та підписує його;
12) метод *transactionExists*(*self, transaction*): перевіряє, чи вже існує така транзакція в блокчейні;
13) метод *forgerValid*(*self, block*): перевіряє, чи правильно вказаний генератор блоку у вхідному блоку *block*;
14) метод *transactionValid*(*self, transactions*): перевіряє, чи можливо виконати всі транзакції з переданого списку.

Файл *AccountModel.py*

```
class AccountModel():
    def __init__(self):
        self.accounts = []
        self.balances = {}
        self.tickets = {}
    def addAccount(self, publicKeyString):
        if not publicKeyString in self.accounts:
            self.accounts.append(publicKeyString)
            self.balances[publicKeyString] = 0
            self.tickets[publicKeyString] = []
    def getBalance(self,publicKeyString):
        if publicKeyString not in self.accounts:
            self.addAccount(publicKeyString)
        return self.balances[publicKeyString]
```

```
    def updateBalace(self,publicKeyString, amount):
        if publicKeyString not in self.accounts:
            self.addAccount(publicKeyString)
        self.balances[publicKeyString] += amount
    def getBalanceTicket(self,publicKeyString):
        if publicKeyString not in self.accounts:
            self.addAccount(publicKeyString)
        return self.tickets[publicKeyString]
    def updateTicket(self,publicKeyString, ticket):
        if publicKeyString not in self.accounts:
            self.addAccount(publicKeyString)
        self.tickets[publicKeyString].append(ticket)
```

Ця програма містить клас *AccountModel*, який представляє модель рахунків у системі. Ось що роблять методи та функції цього класу:

1) конструктор *__init__*(*self*): він ініціалізує атрибути об'єкта *AccountModel*: *accounts* (список публічних ключів облікових записів), *balances* (словник для зберігання балансу кожного облікового запису) та *tickets* (словник для зберігання списку квитків кожного облікового запису);
2) метод *addAccount*(*self, publicKeyString*): додає новий обліковий запис до моделі. Якщо публічний ключ *publicKeyString* ще не існує в *accounts*, він додається, і для нього створюються записи у *balances* та *tickets*;
3) метод *getBalance*(*self, publicKeyString*): повертає баланс для заданого публічного ключа *publicKeyString*. Якщо обліковий запис з цим ключем відсутній, він спочатку додається;
4) метод *updateBalance*(*self, publicKeyString, amount*): оновлює баланс для заданого публічного ключа *publicKeyString* на вказану суму *amount*. Якщо обліковий запис з цим ключем відсутній, він спочатку додається;
5) метод *getBalanceTicket*(*self, publicKeyString*): повертає список квитків для заданого публічного ключа *publicKeyString*. Якщо обліковий запис з цим ключем відсутній, він спочатку додається;

6) метод *updateTicket*(*self, publicKeyString, ticket*): оновлює список квитків для заданого публічного ключа *publicKeyString*, додаючи новий квиток *ticket*. Якщо обліковий запис з цим ключем відсутній, він спочатку додається.

Отже, цей клас *AccountModel* надає методи для додавання, отримання та оновлення облікових записів та їх балансів, а також для керування списком квитків.

Файл *Node.py*

```
class Node():
    def __init__(self, ip, port, key = None):
        self.p2p = None
        self.ip = ip
        self.port = port
        self.transactionPool = TransactionPool()
        self.blockchain = Blockchain()
        self.wallet = Wallet()
        if key is not None:
            self.wallet.fromKey(key)
    def startP2P(self):
        self.p2p = SocketCommunication(self.ip, self.port)
        self.p2p.startSocketCommunication(self)
    def startAPI(self, apiPort):
        self.api = NodeAPI()
        self.api.injectNode(self)
        self.api.start(apiPort)
    def handleTransaction(self, transaction): #broadcast tx through network
        data = transaction.payload()
        signature = transaction.signature
```

```
        signerPublicKey = transaction.senderPublicKey
            signatureValid = Wallet.signatureValid(data, signature,
signerPublicKey)
            transactionExists = self.transactionPool.transactionExists(transaction)
            transactionInBlock = self.blockchain.transactionExists(transaction)
            if not transactionExists and not transactionInBlock and signatureValid:
                self.transactionPool.addTransaction(transaction)
                message = Message(self.p2p.socketConnector, 'TRANSACTION',
transaction)
                encodedMessage = BlockchainUtils.encode(message)
                self.p2p.broadcast(encodedMessage)
                forgingRequired = self.transactionPool.forgerRequired()
                if forgingRequired:
                    self.forge()
            def handleBlock(self,block):
                forger = block.forger
                blockHash = block.payload()
                signature = block.signature
                blockCountValid = self.blockchain.blockCountValid(block)
                lastBlockHashValid = self.blockchain.lashBlockHashValid(block)
                forgerValid = self.blockchain.forgerValid(block)
                transactionsValid =
self.blockchain.transactionValid(block.transactions)
                signatureValid = Wallet.signatureValid(blockHash, signature, forger)
                if not blockCountValid:
                    self.requestChain()
                if lastBlockHashValid and forgerValid and transactionsValid and
signatureValid:
                    self.blockchain.addBlock(block)
                    self.transactionPool.removeFromPool(block.transactions)
```

```
                self.p2p.broadcast(encodedMessage)
            def requestChain(self):
                message = Message(self.p2p.socketConnector,
'BLOCKCHAINREQUEST', None)
                encodedMessage = BlockchainUtils.encode(message)
                self.p2p.broadcast(encodedMessage)
            def handleBlockchainRequest(self, requestingNode):
                message = Message(self.p2p.socketConnector, 'BLOCKCHAIN',
self.blockchain)
                encodedMessage = BlockchainUtils.encode(message)
                self.p2p.send_to_node(requestingNode, encodedMessage)
            def handleBlockchain(self, blockchain):
                localBlockchainCopy = copy.deepcopy(self.blockchain)
                localBlockCount = len(localBlockchainCopy.blocks)
                receivedChainBlockCount = len(blockchain.blocks)
                if localBlockCount < receivedChainBlockCount:
                    for blockNumber, block in enumerate(blockchain.blocks):
                        if blockNumber >= localBlockCount:
                            localBlockchainCopy.addBlock(block)
                            self.transactionPool.removeFromPool(block.transactions)
                    self.blockchain = localBlockchainCopy
            def forge(self):
                forger = self.blockchain.nextForger()
                if forger == self.wallet.publicKeyString():
                    print('im the next forger' )
                    block =
self.blockchain.createBlock(self.transactionPool.transactions, self.wallet) #create
Block
                    self.transactionPool.removeFromPool(block.transactions) #remove
covered tx from pool
```

```
            encodedMessage = BlockchainUtils.encode(message)
            self.p2p.broadcast(encodedMessage) #broadcast block from local to
all network
        else:
            print('im not the next forger')
```

Цей файл містить клас *Node*, який представляє собою вузол у блокчейні. Ось що роблять методи та функції цього класу:

1) конструктор *__init__*(*self, ip, port, key = None*): він ініціалізує атрибути об'єкта *Node*, включаючи *IP*-адресу та порт, об'єкти *TransactionPool*, *Blockchain*, *Wallet*, та, в разі наявності ключа (*key*), ініціалізує його в *Wallet*;
2) метод *startP2P*(*self*): створює об'єкт *SocketCommunication* для забезпечення мережевого зв'язку та запускає його;
3) метод *startAPI*(*self, apiPort*): створює об'єкт *NodeAPI* для обробки *API*-запитів та запускає його;
4) метод *handleTransaction*(*self, transaction*): обробляє отриману транзакцію. Перевіряє її дійсність та додає до пулу транзакцій. Якщо у пулі достатньо транзакцій, викликає метод *forge*() для генерації нового блоку;
5) метод *handleBlock*(*self, block*): обробляє отриманий блок. Перевіряє його дійсність та додає до блокчейну. Видаляє використані транзакції з пулу транзакцій;
6) метод *requestChain*(*self*): відправляє запит на отримання повного ланцюжка блоків;
7) метод *handleBlockchainRequest*(*self, requestingNode*): відправляє повний ланцюжок блоків вузлу, який надіслав запит;
8) метод *handleBlockchain*(*self, blockchain*): обробляє отриманий повний ланцюжок блоків та оновлює локальний блокчейн;
9) метод *forge*(*self*): перевіряє, чи поточний вузол є наступним генератором блоку. Якщо так, генерує новий блок та відправляє його в мережу.

Файл *SocketCommunication.py*

```
class SocketCommunication (Node):
    def __init__(self, ip, port):
        super(SocketCommunication, self).__init__(ip, port, None)
        self.peers = []
        self.PeerDiscoveryHandler = PeerDiscoveryHandler(self)
        self.socketConnector = SocketConnector(ip, port)
    def connectToFirstNode(self):
        if self.socketConnector.port != 10001:
            self.connect_with_node('localhost',10001)
    def startSocketCommunication(self, node):
        self.node = node
        self.start()
        self.PeerDiscoveryHandler.start()
        self.connectToFirstNode()
    def inbound_node_connected(self, connected_node):
        self.PeerDiscoveryHandler.handshake(connected_node)
    def outbound_node_connected(self, connected_node):
        self.PeerDiscoveryHandler.handshake(connected_node)
    def node_message(self, connected_node, message):
        message = BlockchainUtils.decode(json.dumps(message))
        if message.messageType == 'DISCOVERY':
            self.PeerDiscoveryHandler.handleMessage(message)
        elif message.messageType == 'TRANSACTION':
            transaction = message.data
            self.node.handleTransaction(transaction)
        elif message.messageType == 'BLOCK':
            block = message.data
            self.node.handleBlock(block)
        elif message.messageType == 'BLOCKCHAINREQUEST':
```

```
self.node.handleBlockchainRequest(connected_node)
        elif message.messageType == 'BLOCKCHAIN':
            blockchain = message.data
            self.node.handleBlockchain(blockchain)
    def send(self, receiver, message):
        self.send_to_node(receiver,message)
    def broadcast(self, message): #need if fail connection for first try
        self.send_to_nodes(message)
```

Цей файл реалізує комунікацію між вузлами мережі за допомогою сокетів і використовує бібліотеку *p2pnetwork*. Ось що роблять методи та функції цього класу:

1) конструктор *__init__*(*self, ip, port*): його основне призначення - налаштувати параметри мережевого з'єднання та ініціалізувати об'єкт *PeerDiscoveryHandler* та *SocketConnector*;
2) метод *connectToFirstNode*(*self*): намагається підключитися до першого вузла у мережі;
3) метод *startSocketCommunication*(*self, node*): запуск мережевого з'єднання та обробки повідомлень вузлів;
4) метод *inbound_node_connected*(*self, connected_node*): викликається, коли вузол встановлює вхідне з'єднання з іншим вузлом. Викликає *handshake* для обміну даними;
5) метод *outbound_node_connected*(*self, connected_node*): викликається, коли вузол встановлює вихідне з'єднання з іншим вузлом. Викликає *handshake* для обміну даними;
6) метод *node_message*(*self, connected_node, message*): обробка отриманого повідомлення від іншого вузла. Відправляє отримане повідомлення на відповідну обробку вузлу;
7) метод *send*(*self, receiver, message*): відправка повідомлення конкретному вузлу;
8) метод *broadcast*(*self, message*): відправка повідомлення всім вузлам у мережі.

Файл *Message.py*

```
class Message():
    def __init__(self, senderConnector, messageType, data):
        self.senderConnector = senderConnector
        self.messageType = messageType
        self.data = data
```

Цей файл містить опис класу *Message*, який використовується для створення, представлення та передачі повідомлень у системі програмі. Він містить конструктор *__init__*(*self, senderConnector, messageType, data*). Його основне призначення - ініціалізувати атрибути об'єкта *Message*:

1) *senderConnector*: представляє з'єднання відправника повідомлення. Це може бути, наприклад, об'єкт, який представляє з'єднання з сокетом або іншим мережевим засобом;
2) *messageType*: вказує на тип повідомлення (наприклад, '*TRANSACTION*', '*BLOCK*', тощо);
3) *data*: містить в собі конкретні дані або об'єкт, який відображається у повідомленні.

Цей клас *Message* слугує для створення та передачі повідомлень у вашій програмі, і його атрибути дозволяють вказати відправника, тип повідомлення та конкретні дані, які передаються разом з повідомленням.

Файл *SocketConnector.py*

```
class SocketConnector():
    def __init__(self, ip, port):
        self.ip = ip
        self.port = port

    def equals(self, connector):
        if connector.ip == self.ip and connector.port == self.port:
            return True
```

```
        else:
            return False
```

Цей файл містить клас *SocketConnector*, який використовується для представлення з'єднання з сокетом у програмі. Ось що роблять методи та атрибути цього класу:

1) конструктор *__init__*(*self, ip, port*): його основне призначення - ініціалізувати атрибути об'єкта *SocketConnector*:
   - *ip*: Представляє *IP*-адресу для підключення;
   - *port*: Представляє порт для підключення;
2) метод *equals*(*self, connector*): порівнює об'єкт *SocketConnector* з іншим об'єктом *connector* і повертає *True*, якщо їх *IP* та порт співпадають, і *False* у протилежному випадку.

Цей клас *SocketConnector* слугує для представлення і порівняння з'єднань з сокетами. Ймовірно, він використовується в вашій програмі для визначення адрес та портів вузлів або пірів у мережі.

Файл *PeerDiscoveryHandler.py*

```
class PeerDiscoveryHandler():
    def __init__(self, node):
        self.socketCommunication = node
    def start(self):
        statusThread = threading.Thread(target=self.status,args=())
        statusThread.start()
        discoveryThread = threading.Thread(target=self.discovery,args=())
        discoveryThread.start()
    def status(self):
        while True:
            print('Current Connections:')
            for peer in self.socketCommunication.peers:
                print(str(peer.ip) + ':' + str(peer.port))
```

```
        time.sleep(5)
def discovery(self):
     while True:
        handshakeMessage = self.handshakeMessage()
        self.socketCommunication.broadcast(handshakeMessage)
        time.sleep(10)
def handshake(self, connect_node): #exchange info between nodes
    handshakeMessage = self.handshakeMessage()
    self.socketCommunication.send(connect_node, handshakeMessage)
def handshakeMessage(self):
    ownConnector = self.socketCommunication.socketConnector
    ownPeers = self.socketCommunication.peers
    data = ownPeers
    messageType = 'DISCOVERY'
    message = Message(ownConnector, messageType, data)
    encodedMessage = BlockchainUtils.encode(message)
    return encodedMessage
def handleMessage(self, message):
    peersSocketConnector = message.senderConnector
    peersPeerList = message.data
    newPeer = True
    for peer in self.socketCommunication.peers:
        if peer.equals(peersSocketConnector):
            newPeer = False
    if newPeer:

        self.socketCommunication.peers.append(peersSocketConnector)
    for peersPeer in peersPeerList:
        peerKnown = False
        for peer in self.socketCommunication.peers:
```

```
            if peer.equals(peersPeer):
                peerKnown = True
        if not peerKnown and not
peersPeer.equals(self.socketCommunication.socketConnector):
            self.socketCommunication.connect_with_node(peersPeer.ip,
peersPeer.port)
```

Цей файл містить клас *PeerDiscoveryHandler*, який відповідає за виявлення та обмін інформацією між вузлами мережі. Ось що роблять методи та атрибути цього класу:

1) конструктор *__init__*(*self, node*): його основне призначення - ініціалізувати атрибут *socketCommunication* об'єкта *node*, який представляє собою комунікаційний канал з вузлами мережі;
2) метод *start*(*self*): для початку роботи потоків *statusThread* і *discoveryThread*;
3) метод *status*(*self*): виводить поточний статус підключених вузлів. Виводить *IP* та порт кожного вузла кожні 5 секунд;
4) метод *discovery*(*self*): відправляє повідомлення рукостискання (*handshakeMessage*) всім вузлам у мережі кожні 10 секунд;
5) метод *handshake*(*self, connect_node*): відправляє повідомлення рукостискання конкретному вузлу (*connect_node*);
6) метод *handshakeMessage*(*self*): створює повідомлення рукостискання;
7) метод *handleMessage*(*self, message*): для обробки отриманого повідомлення. Отримує інформацію про вузли, які відправили повідомлення, та оновлює список вузлів мережі;
8) метод *handle*(*self, connect_node*): обробляє нове підключення до вузла.

Файл *NodeAPI.py*

```
class NodeAPI(FlaskView):
   def __init__(self):
      self.app = Flask(__name__)  #flask app
      self.app.secret_key = 'supersecretkey1'
```

```
    def start(self, apiPort):
        NodeAPI.register(self.app, route_base='/')
        self.app.run(host = 'localhost', port = apiPort)
    def injectNode(self, injectednode):
        global node
        node = injectednode
```

Цей файл містить клас *NodeAPI*, який відповідає за взаємодію з вузлом через *API* (призначене для зв'язку між клієнтом і сервером).

Давайте розглянемо деякі основні методи та їх функціональність:

1) метод *start*(*self, apiPort*) для запуску *Flask*-додатку з *API* за певним портом *apiPort*;
2) метод *injectNode*(*self, injectednode*) для вставки посилання на вузол у клас *NodeAPI*. В цьому випадку, посилання на вузол *node* ініціалізується;
3) маршрути, які пов'язані з вебсторінками, отримують дані від користувача, виконують деякі дії та повертають відповідь на клієнтську сторону (*HTML*-сторінки). Наприклад: /*blockchain* - повертає *JSON*-представлення блокчейну;
4) та інші, які обробляють введені дані та повертають результати користувачеві;
5) маршрути, які служать для переадресації та переходу між сторінками вебсайту;
6) велика кількість методів для роботи з різними сторінками вебсайту та виконання певних дій. Наприклад, відображення сторінки після успішного створення гаманця, додавання коштів на баланс, резервації квитків тощо;
7) велика кількість методів, які служать для переадресації на інші сторінки сайту;
8) деякі методи виконують роботу з транзакціями, балансами та іншими даними, використовуючи вузол.

Файл *ProofOfStake.py*

```
class ProofOfStake():
    def __init__(self):
        self.stackers = {}
        self.setGenesisNodeStake()
    def setGenesisNodeStake(self):
        genesisPublicKey = open('keys/genesisPublicKey.pem', 'r').read()
        self.stackers[genesisPublicKey] = 1
    def update(self, publicKeyString, stake):
        if publicKeyString in self.stackers.keys():
            self.stackers[publicKeyString] += stake
        else:
            self.stackers[publicKeyString] = stake

    def get(self, publicKeyString):
        if publicKeyString in self.stackers.keys():
            return self.stackers[publicKeyString]
        else:
            return None

    def validatorLots(self, seed):
        lots = []
        for validator in self.stackers.keys():
            for stake in range(self.get(validator)):
                lots.append(Lot(validator, stake + 1, seed))
        return lots

    def winnerLot(self, lots, seed):
        winnerLot = None
        leastOffset = None
        referenceHashIntValue = int(BlockchainUtils.hash(seed).hexdigest(),
16)
```

```
        for lot in lots:
            lotIntValue = int(lot.lotHash(), 16)
            offset = abs(lotIntValue - referenceHashIntValue)
            if leastOffset is None or offset < leastOffset:
                leastOffset = offset
                winnerLot = lot
        return winnerLot
    def forger(self, lastBlockHash):
        lots = self.validatorLots(lastBlockHash)
        winnerLot = self.winnerLot(lots, lastBlockHash)
        return winnerLot.publicKey
```

Цей файл містить клас *ProofOfStake*, який відповідає за реалізацію алгоритму *Proof of Stake* (*PoS*) в системі блокчейну. Основні функції та їх функціональність:

1) конструктор*__init__*(*self*): ініціалізує об'єкт *PoS* з порожнім словником *stackers* та встановлює ставку для генезис-вузла;
2) метод *setGenesisNodeStake*(*self*): встановлює ставку для генезис-вузла, яка зберігається в *stackers*;
3) метод *update*(*self, publicKeyString, stake*): оновлює ставку для заданого публічного ключа. Якщо ключ вже існує, то ставка збільшується на задану величину *stake*;
4) метод *get*(*self, publicKeyString*): повертає ставку для заданого публічного ключа, якщо ключ існує, інакше повертає *None*;
5) метод *validatorLots*(*self, seed*): генерує список лотів для всіх валідаторів на основі вказаного насіння (*seed*). Кількість лотів для кожного валідатора визначається його ставкою;
6) метод *winnerLot*(*self, lots, seed*): обирає переможний лот серед набору лотів *lots* на основі вказаного насіння (*seed*). Визначається лот з найменшим відхиленням від відомостей *seed*;

7) метод *forger*(*self, lastBlockHash*): визначає вузол, який має право форжити новий блок. Для цього генерується набір лотів на основі *lastBlockHash* і обирається переможець.

Цей клас реалізує алгоритм *PoS*, що використовується для вибору користувача, який буде створювати новий блок в мережі блокчейну.

Файл *Lot.py*

```
class Lot():
    def __init__(self, publicKey, iteration, lastBlockHash):
        self.publicKey = str(publicKey)
        self.iteration = iteration
        self.lastBlockHash = lastBlockHash
    def lotHash(self):
        hashData = self.publicKey + self.lastBlockHash
        for _ in range(self.iteration):
            hashData = BlockchainUtils.hash(hashData).hexdigest()
        return hashData
```

Цей файл містить клас *Lot*, який представляє об'єкт лоту для алгоритму *Proof of Stake* (*PoS*). В основному, цей клас використовується в контексті визначення переможця, який має право створювати новий блок.

Основні методи та їх функціональність:

1) конструктор *__init__*(*self, publicKey, iteration, lastBlockHash*): він ініціалізує об'єкт лоту з вказаними параметрами: *publicKey* - публічний ключ, *iteration* - номер ітерації та *lastBlockHash* - хеш останнього блоку;
2) метод *lotHash*(*self*): обчислює хеш лоту на основі публічного ключа, хешу останнього блоку та номера ітерації. У цьому методі використовується кілька повторень виклику хеш-функції для підвищення варіативності результату.

Отже, об'єкт класу *Lot* є представленням конкретного лоту для валідатора в алгоритмі *PoS*. Цей лот використовується для визначення переможця, який отримує право створювати новий блок у мережі блокчейну.

3.2. Інтеграція функціоналу для купівлі авіаквитків в блокчейн

Файл *Airplane.py*

```
class Airplane():
    def __init__(self):
        self.fromWhere = "Київ"
        self.where = "Львів"
        self.date = "25.12.2023"
        self.timeDeparture = "9:00"
        self.timeArrival = "12:00"
        self.companyName = "МАУ"
        self.tickets = [Ticket() for _ in range(20)]

    def printTickets(self):
        for i in range (len(self.tickets)):
            print(i)
            print(self.tickets[i].price)
            print(self.tickets[i].ifbuy)
            print(self.tickets[i].id)

    def payload(self):
        jsonRepresentation = copy.deepcopy(self.toJson())
        jsonticket = []
        for ticket in self.tickets:
          jsonticket.append(ticket.toJson())
        jsonRepresentation['tickets'] = jsonticket
        jsonRepresentation['signature'] = ''
        return jsonRepresentation
```

```
    def toJson(self):
        return self.__dict__
    def addAirplane(self):
        self.tickets = [Ticket() for _ in range(20)]
```

Цей файл містить клас *Airplane*, який представляє об'єкт «Літак». Основна його мета - управління квитками на рейс.

Основні методи та їх функціональність:

1) конструктор *__init__*(*self*): він ініціалізує атрибути, такі як місце відправлення, місце прибуття, дата вильоту, час вильоту, час прибуття, назва авіакомпанії та генерує список квитків;
2) метод *printTickets*(*self*): виводить інформацію про квитки на екран (ціну, стан купівлі та ідентифікатор);
3) метод *payload*(*self*): повертає об'єкт у форматі *JSON*, включаючи всі квитки;
4) метод *toJson*(*self*): повертає атрибути об'єкта у форматі словника;
5) метод *addAirplane*(*self*): додає новий набір квитків до літака.

Файл *Ticket.py*

```
class Ticket():
    def __init__(self):
        self.price = 10
        self.ifbuy = False
        self.id = uuid.uuid4()
    def toJson(self):
        return self.__dict__
    def buy(self):
        self.ifbuy = True
```

Цей файл містить клас *Ticket*, який представляє квиток.

Основні методи та їх функціональність:

1) конструктор *__init__*(*self*): він ініціалізує атрибути квитка, такі як ціна, стан купівлі (*ifbuy*) та унікальний ідентифікатор (*id*) за допомогою модуля *uuid*;

2) метод *toJson*(*self*): повертає атрибути об'єкта у форматі словника;

3) метод *buy*(*self*): встановлює стан квитка як куплений (*ifbuy = True*).

Файл *BlockchainUtils.py*

```
class BlockchainUtils():
    @staticmethod
    def hash(data):
        dataString = json.dumps(data)
        dataBytes = dataString.encode('utf-8')
        dataHash = SHA256.new(dataBytes)
        return dataHash
    @staticmethod
    def encode(objectToEncode):
        return jsonpickle.encode(objectToEncode,unpicklable=True)
    @staticmethod
    def decode(encodedObject):
        return jsonpickle.decode(encodedObject)
```

Цей файл містить декілька корисних функцій, які допомагають в роботі з блокчейном:

1) метод *hash*(*data*): генерує хеш для наданого об'єкта *data*. Використовує модуль *SHA*256 з бібліотеки *Cryptodome*. Оперує в форматі *UTF*-8;

2) метод *encode*(*objectToEncode*): кодує наданий об'єкт *objectToEncode* в *JSON*-формат за допомогою бібліотеки *jsonpickle*;

3) метод *decode*(*encodedObject*): розкодовує *JSON*-об'єкт, відкодований за допомогою *jsonpickle*.

Файл *Main.py*

```
if __name__ == '__main__':
    ip = sys.argv[1]
    port = int(sys.argv[2])
    apiPort = int(sys.argv[3])
```

```
keyFile = None
if len(sys.argv) > 4:
    keyFile = sys.argv[4]
node = Node(ip, port, keyFile)
node.startP2P()
node.startAPI(apiPort)
```

Цей файл має на меті створити екземпляр вузла (*Node*) і запустити його:

1) зчитує вхідні аргументи командного рядка:
   - *ip*: *IP*-адреса, за якою вузол буде доступний у мережі;
   - *port*: порт для з'єднання з іншими вузлами;
   - *apiPort*: порт для запуску *API* для взаємодії з вузлом;
   - *keyFile* (необов'язковий): шлях до файлу з ключами, якщо вони передаються в командному рядку;
2) створює новий екземпляр вузла (*Node*) з вказаними параметрами;
3) запускає *P2P* комунікації для обміну даними з іншими вузлами;
4) запускає *API* для взаємодії з вузлом через *HTTP*-запити.

3.3. Висновки до розділу

У даному розділі надано огляд проектування та розробки блокчейну для купівлі авіаквитків. В основі цього рішення лежить алгоритм консенсусу *Proof of Stake* (*PoS*) зі швидкістю обробки транзакцій до 3 секунд. Це дозволяє нам забезпечити високу ефективність та швидкість обслуговування.

Ключовим аспектом є те, що самі квитки зберігаються в блокчейні. Це усуває потенційні ризики шахрайства та вразливостей, які часто виникають при використанні смарт-контрактів. Кожен квиток представлений у вигляді унікального токену з власним унікальним ідентифікатором, що гарантує його аутентичність та унікальність.

Розроблена програма включає різні класи та файли, які спільно реалізують блокчейн та його функціональність. Основні компоненти включають алгоритм *Proof*

of Stake (*PoS*), який використовується для створення нових блоків, управління квитками на літаки та забезпечення комунікації між вузлами мережі.

Файли *ProofOfStake.py* та *Lot.py* відповідають за алгоритм *PoS*, де кожен учасник може внести свої токени та забезпечити ставку. Це визначає, хто матиме право створити новий блок та отримати нагороду.

Файли *Airplane.py* та *Ticket.py* використовуються для керування квитками на літаки, включаючи їх створення, купівлю та відображення.

Файл *BlockchainUtils.py* містить корисні функції для роботи з блокчейном, такі як хешування та кодування, або декодування об'єктів.

У цілому, моя програмна система представляє собою комплексне рішення, що об'єднує аспекти блокчейну, управління квитками на літаки та мережеву комунікацію між вузлами.

РОЗДІЛ 4
ПРИКЛАДИ ЗАСТОСУВАННЯ

4.1. Головна сторінка вебінтерфейсу користувача

Сторінка з якої починається взаємодія користувача з програмною системою у верхній частині екрану вона містить перелік кнопок, за допомогою яких можна перейти до бронювання, реєстрації, входу або переглянути інформацію про авіакомпанію та контакти (див. рис. 4.1).

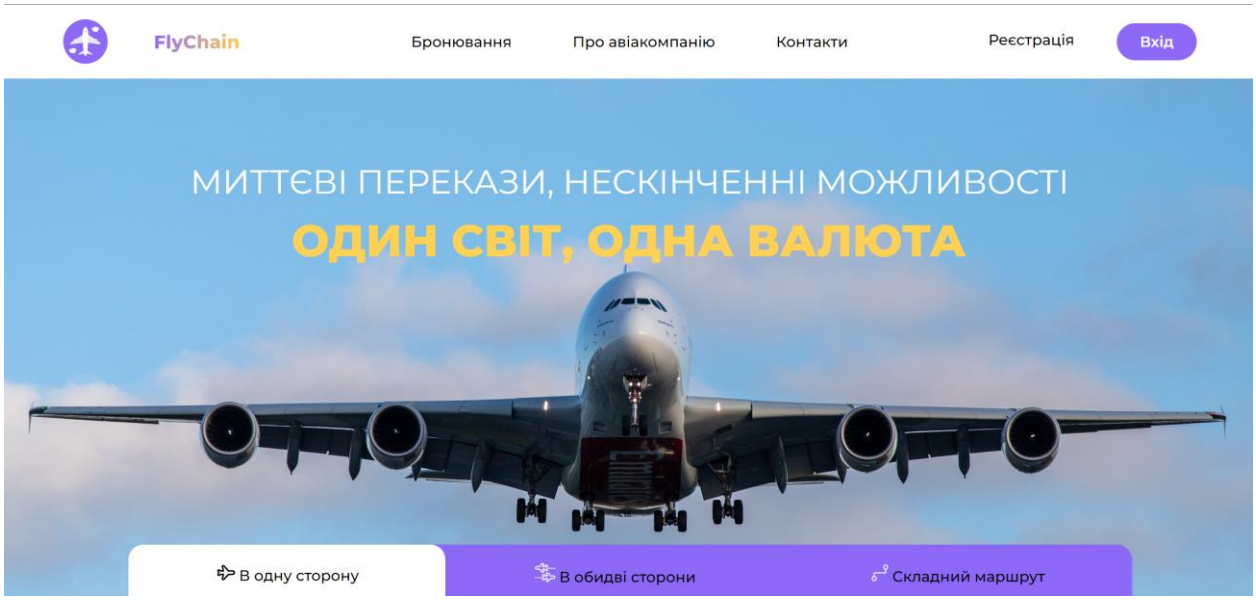

Рис. 4.1. Верхня частина головної сторінки

Користувач може також скористатись панеллю пошуку обравши одну з опцій відповідно до необхідності – «В одну сторону», «В обидві сторони», «Складний маршрут». Після чого вказати відповідну інформацію у поля та натиснути кнопку пошуку (див. рис. 4.2).

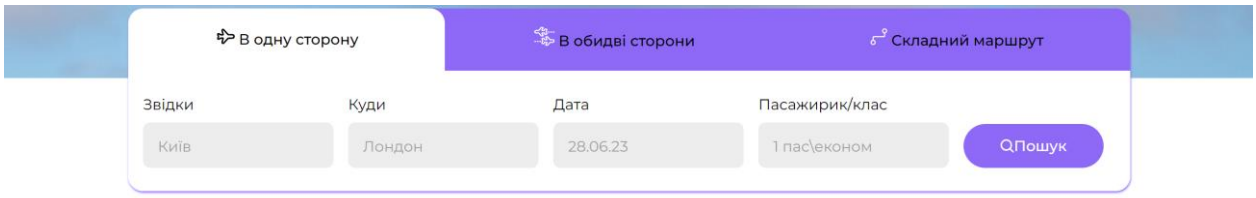

Рис. 4.2. Панель пошуку

Окрім того, на головній сторінці можна ознайомитись з популярними напрямками (див. рис. 4.3).

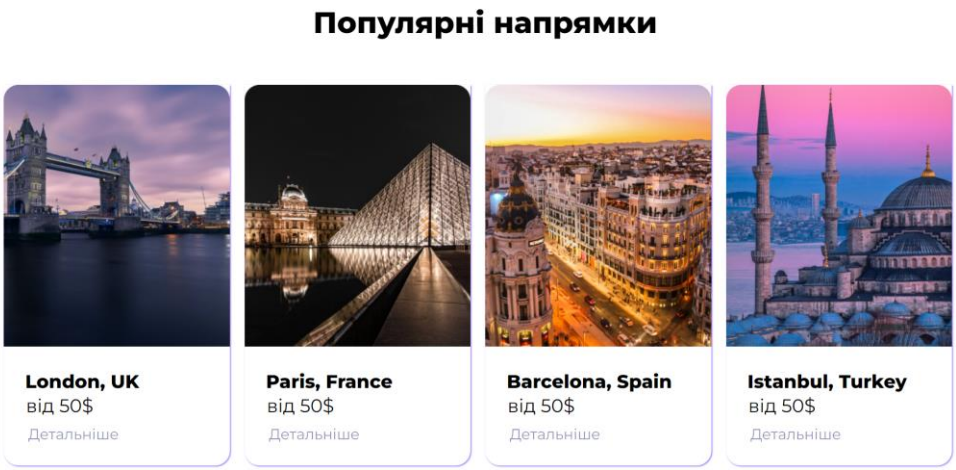

Рис. 4.3. Популярні напрямки

Користувач може ознайомитись з перевагами компанії, яка використовує блокчейн (див. рис. 4.4).

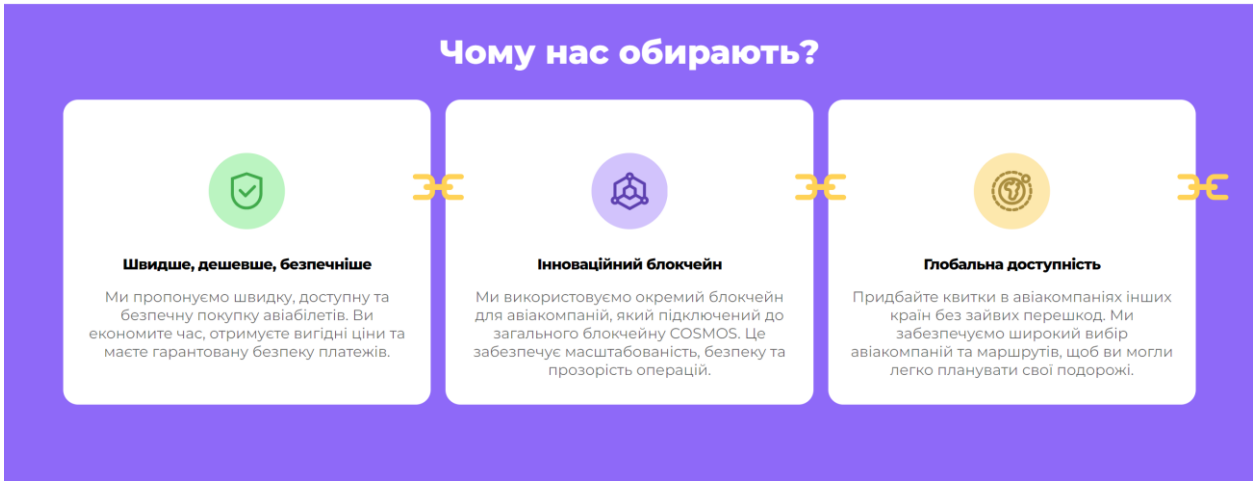

Рис. 4.4. Переваги компанії

В нижній частині головної сторінки є інформацію про внутрішню валюти блокчейну. Та коротка інструкція зі створення власного крипто-гаманця для купівлі авіаквитків (див. рис. 4.5).

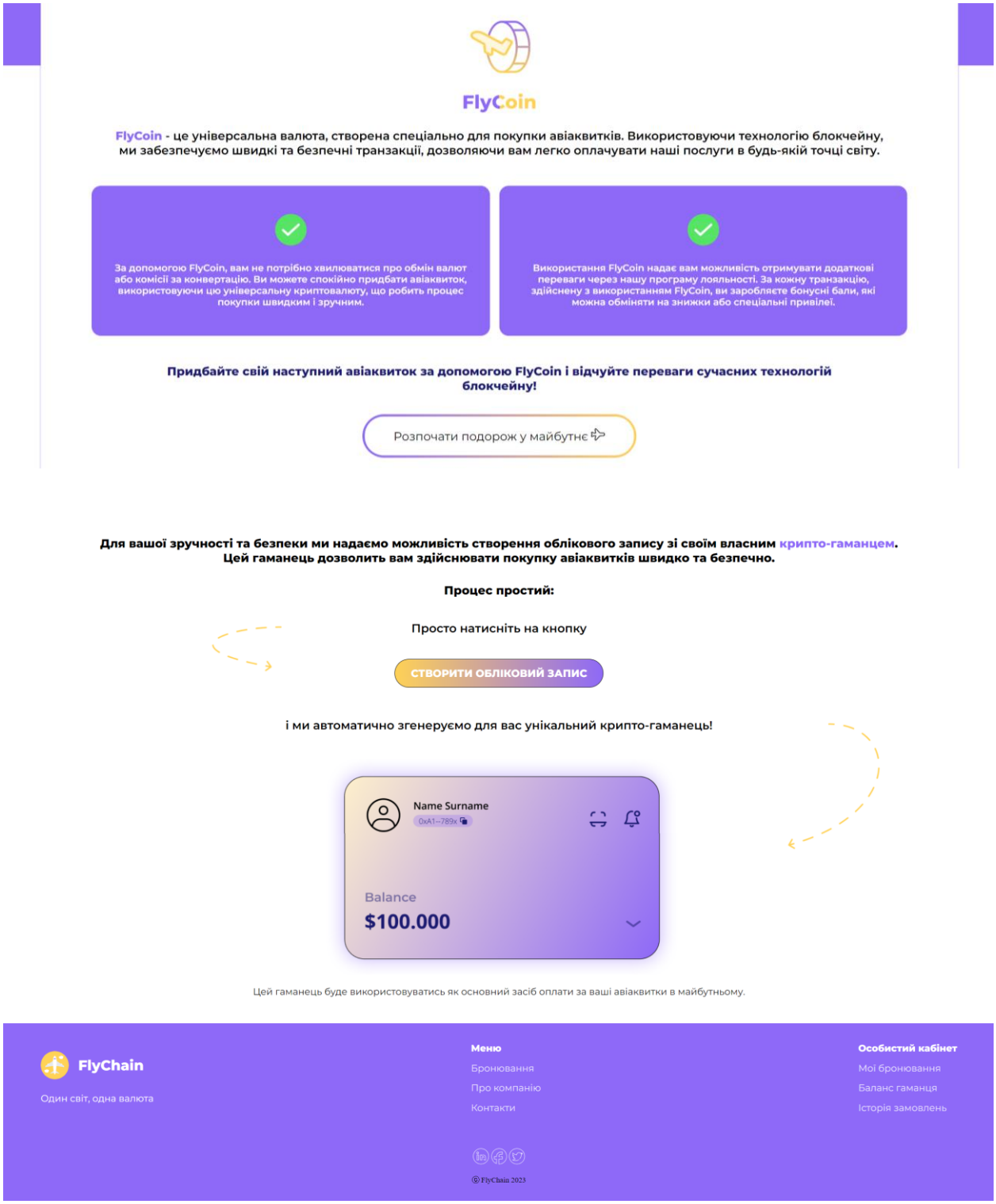

Рис. 4.5. Нижня частина сторінки

4.2. Процес реєстрації та входу до особистого кабінету

Сторінка для реєстрації, яка містить лише одну форму з кнопкою. Після натискання користувач потрапляє на наступний крок – збереження ключів (див. рис. 4.6).

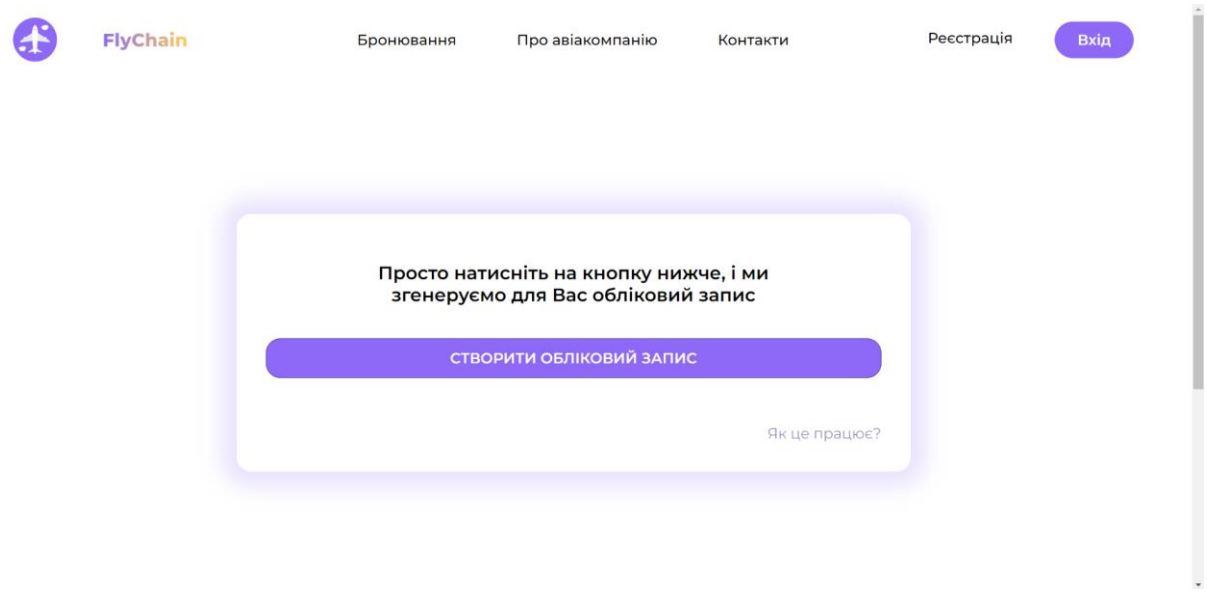

Рис. 4.6. Кнопка реєстрації

Сторінка на якій виводяться ключі для їх збереження. В подальшому ці ключі будуть потрібні для авторизації. Після натискання на кнопку «Продовжити», користувач потрапляє до особистого кабінету (див. рис. 4.7, 4.8).

Рис. 4.7. Форма для збереження ключів

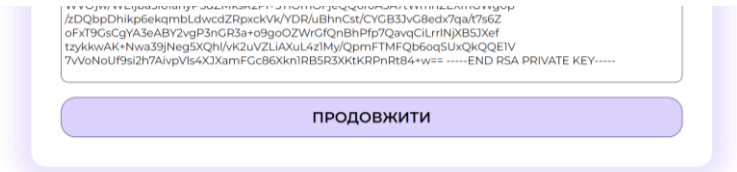

Рис. 4.8. Кнопка «Продовжити»

Сторінка для входу в особистий кабінет, використовуючи попередньо збережені ключі. Після натискання на кнопку «Увійти» користувач потрапляє до особистого кабінету. Якщо користувач натискає на кнопку «Зареєструватись» потрапляє до сторінки зі створенням облікового запису (див. рис. 4.9).

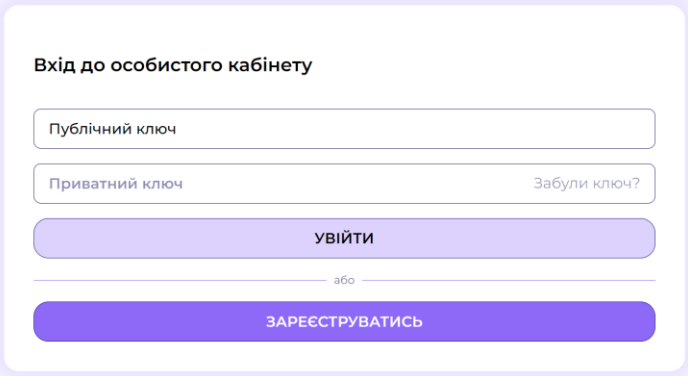

Рис. 4.9. Форма для входу в особистий кабінет

Сторінка з особистого кабінету (див. рис. 4.10), вона містить додаткові вкладки - «Архів подорожей», «Мій крипто-гаманець», «Повідомелння про ціну», «Додати друга», «Налаштування». У вкладці «Мій крипто-гаманець» міститься інформація про актуальні подорожі, ім’я і прізвище користувача, адресу крипто-гаманця та баланс (який можна поповнити натиснувши відповідну кнопку)

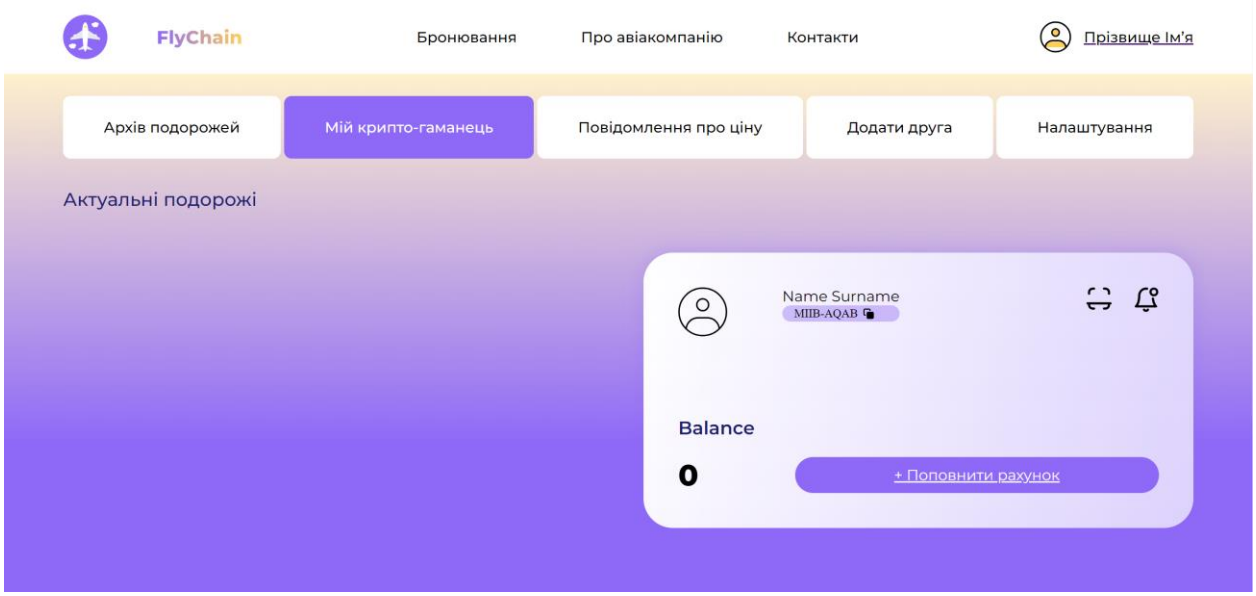

Рис. 4.10. Особистний кабінет

Після поповнення баланс автоматично оновлюється, оскільки дані підтягуються одразу з блокчейну (див. рис. 4.11).

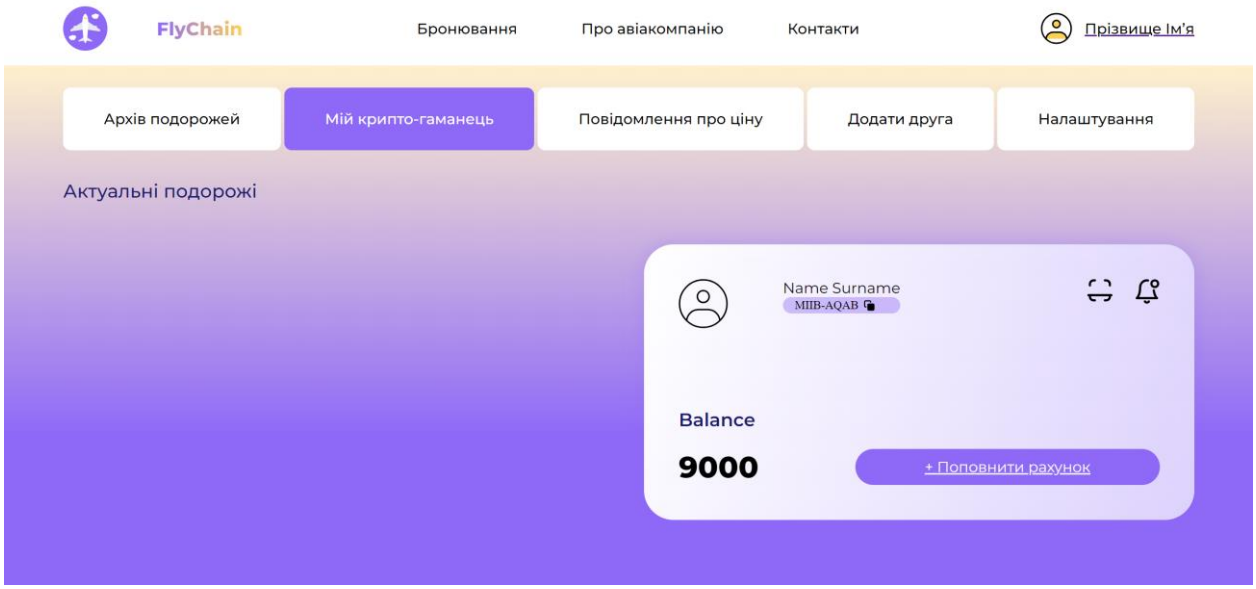

Рис. 4.11. Особистий кабінет після поповнення

4.3. Процес купівлі авіаквитків

Процес купівлі квитків проходить в декілька кроків (див. рис. 4.12). Спочатку користувач вводить всі потрібні йому дані для підбору квитка в системі. Після того як квиток було обрано, потрібно натиснути на кнопку «Обрати квиток». Всі дані про квитки виводиться з блокчейну.

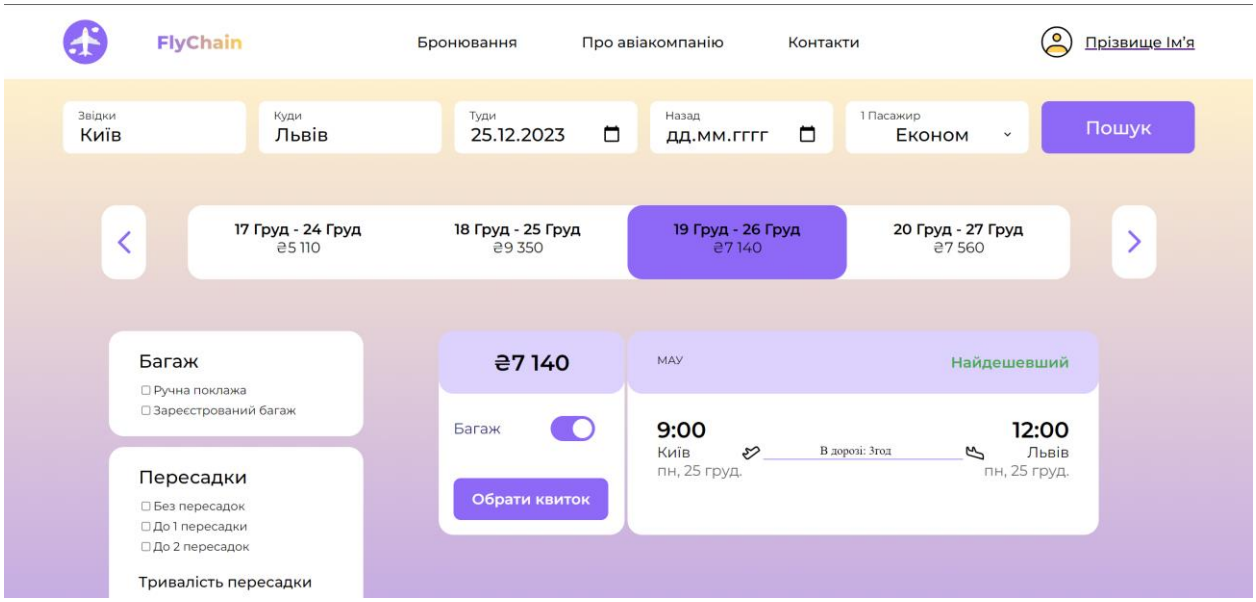

Рис. 4.12. Сторінка пошуку квитків

Користувач потрапляє до вкладки «Інформація про білет», де може ще раз ознайомитись з інформацією про квиток і впевнитись у вірності даних (див. рис. 4.13).

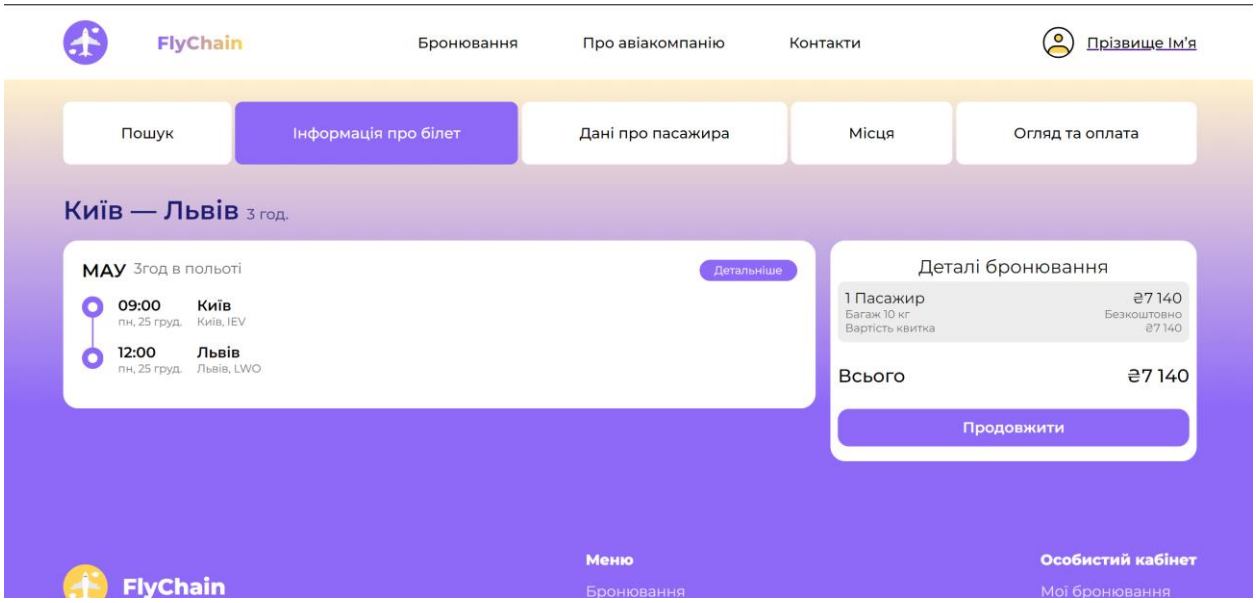

Рис. 4.13. Сторінка інформації про білет

Далі пасажиру потрібно ввести інформацію про себе, при необхідності іншого пасажира, у форму (див. рис. 4.15).

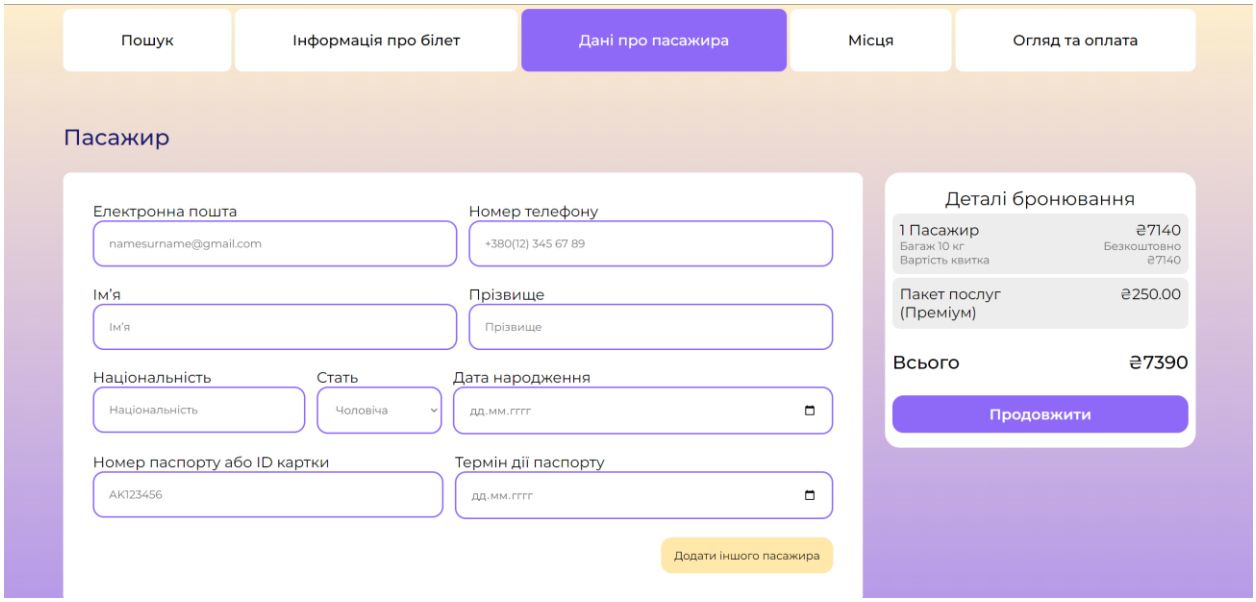

Рис. 4.15. Форма з даними про пасажира

У цій вкладці обрати пакет послуг (див. рис. 4.16).

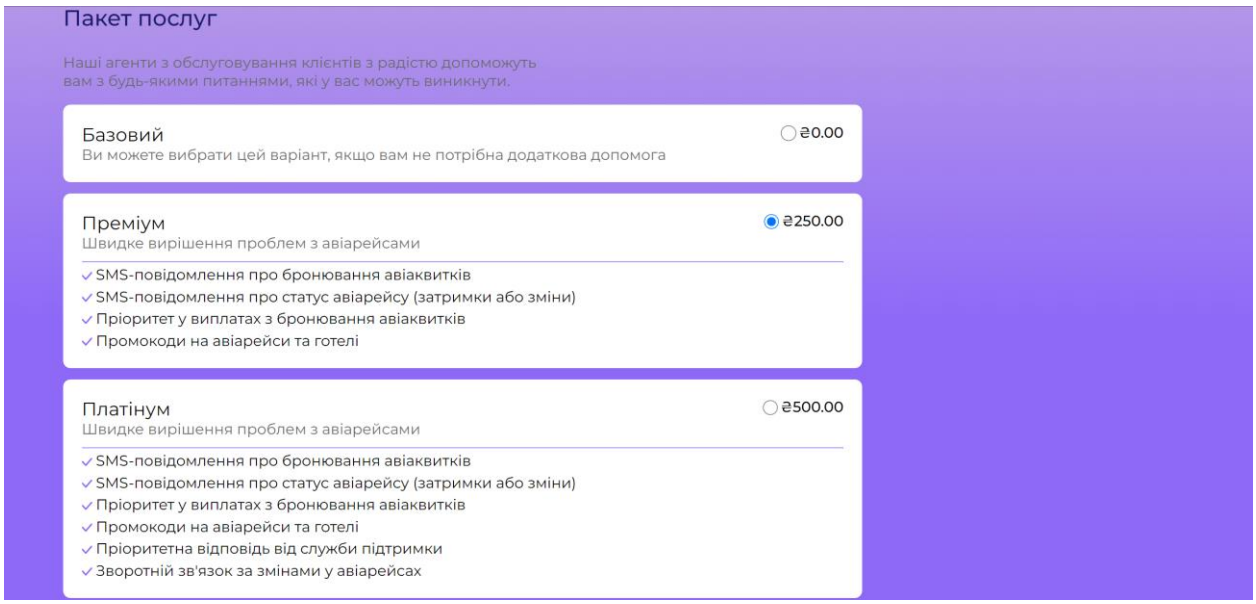

Рис. 4.16. Форма вибору пакету послуг

Після введення всіх даних користувачу слід натиснути на кнопку «Продовжити» (див. рис. 4.17).

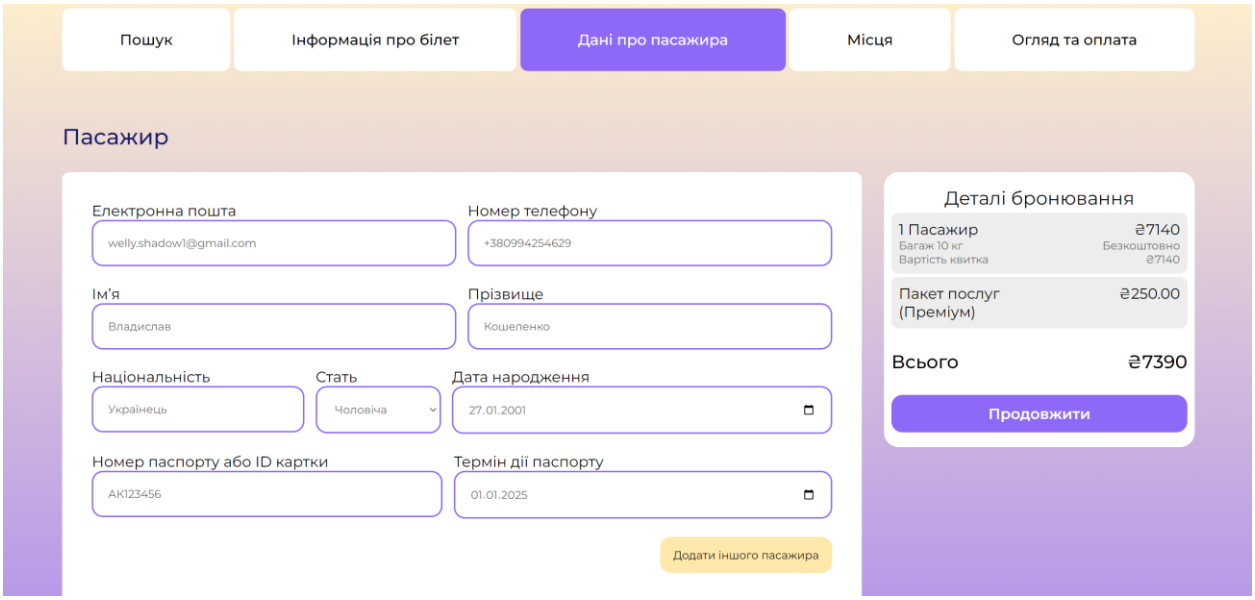

Рис. 4.17. Форма з введеними даними про пасажира

Сторінка «Місця», яка містить інфомацію про всі місця в літаку. Всі дані також виводяться з блокчейну (див. рис. 4.18).

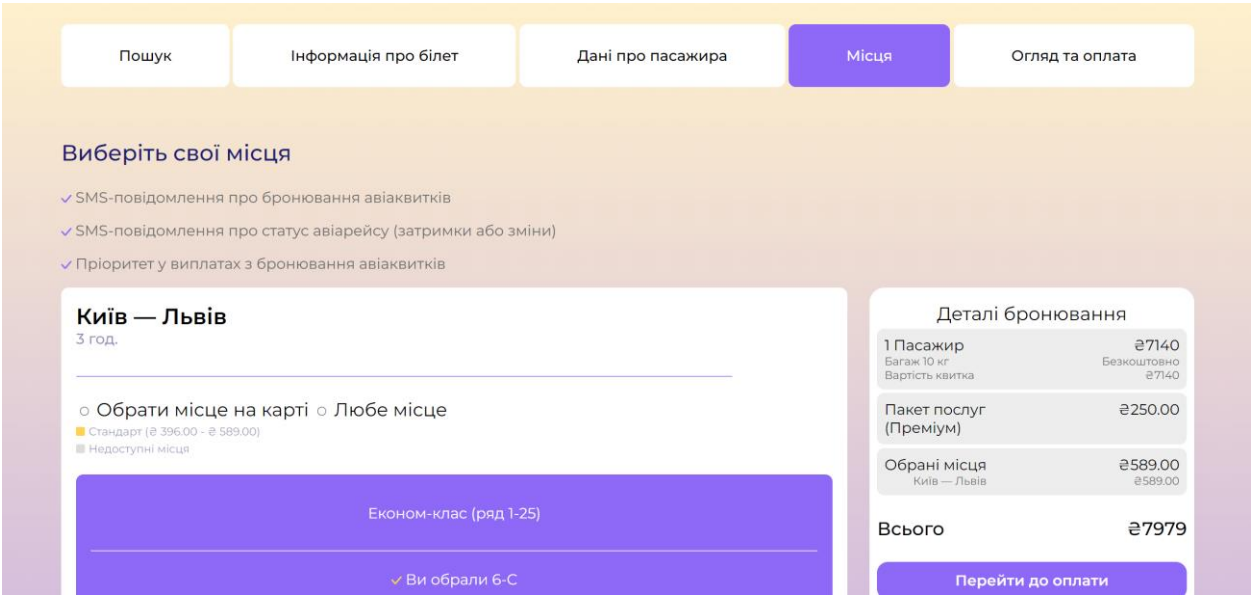

Рис. 4.18. Сторінка з вибором місця

Користувачу слід обрати місце, після чого натиснути на кнопку «Перейти до оплати» (див. рис. 4.19).

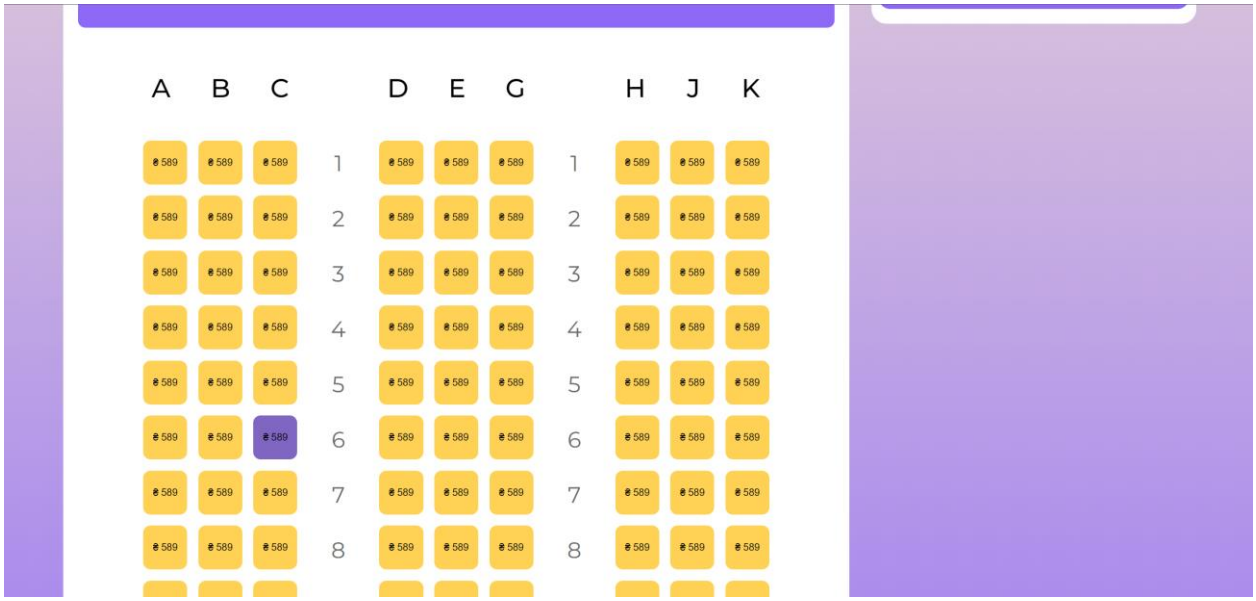

Рис. 4.19. Форма з вибором місця в літаку

На сторінці «Огляд і оплата» пасажир ще раз може ознайомитись з усією інформацією та перевірити її (див. рис. 4.20).

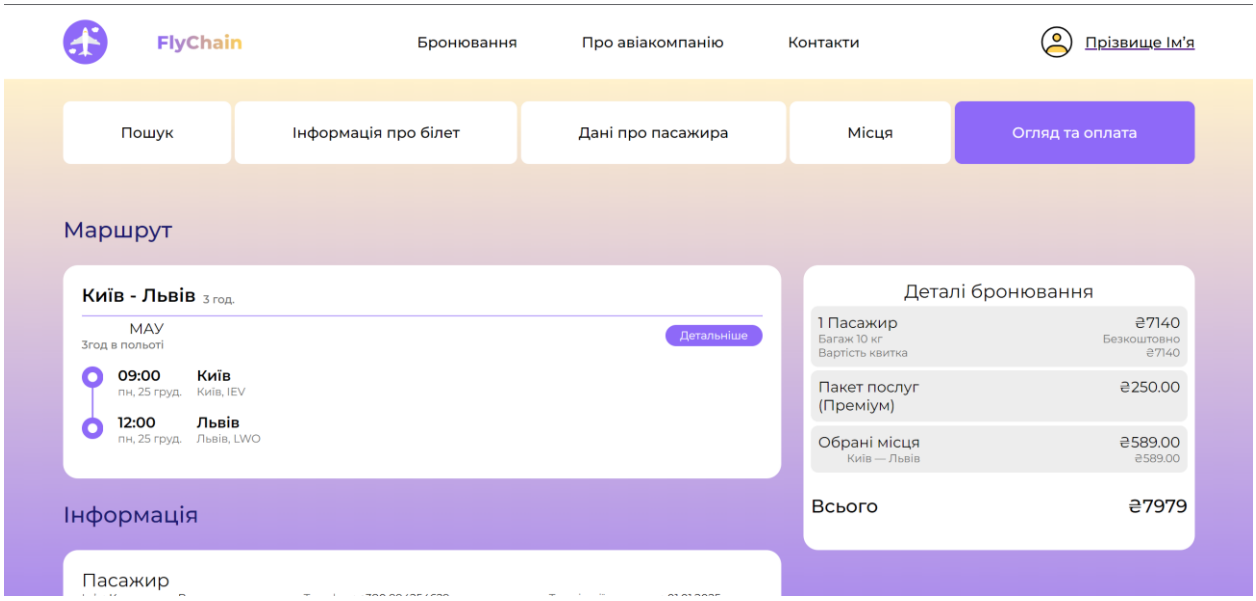

Рис. 4.20. Сторінка огляду інформації про квиток

Якщо користувач перевірив всі дані, та готовий зробити покупку, йому слід натиснути кнопку «Сплатити» (див. рис. 4.21).

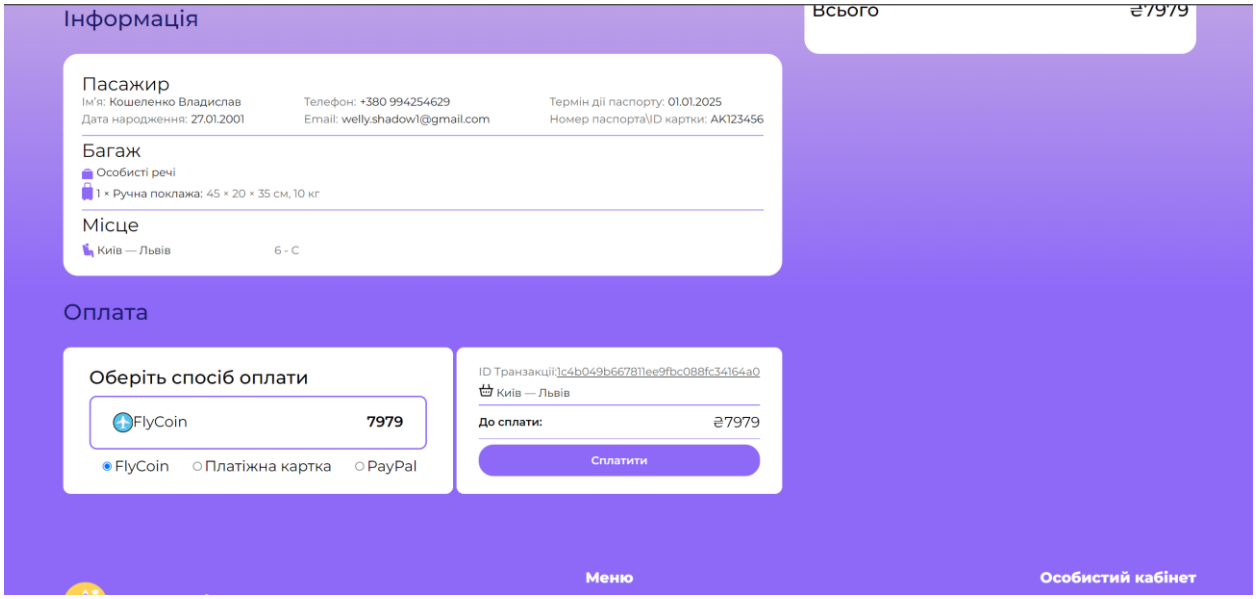

Рис. 4.21. Сплата за квиток

В кінці процесу купівлі користувач отримає форму з даними по транзакції та її підтвердженням, дані також виводяться з блокчейну (див. рис. 4.22).

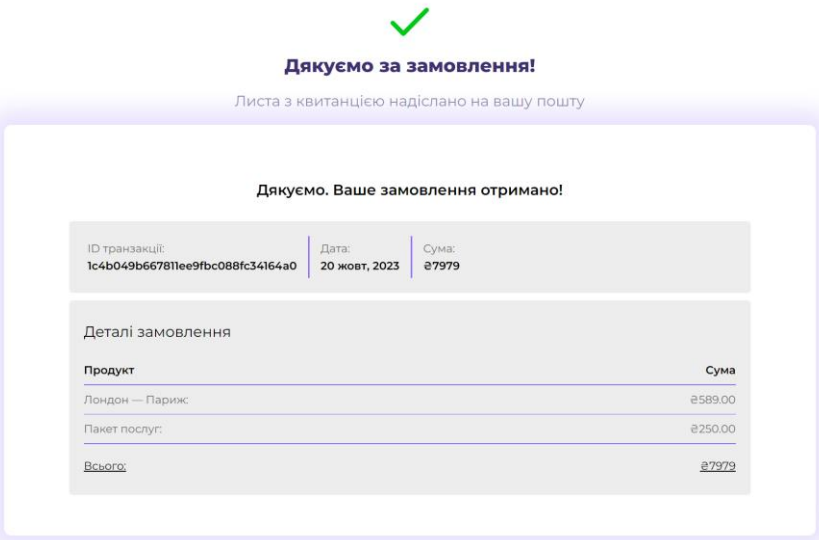

Рис. 4.22. Форма з підтверженям оплати

Після покупки авіаквитка, він відображається у власному кабінеті користувача (див. рис. 4.23).

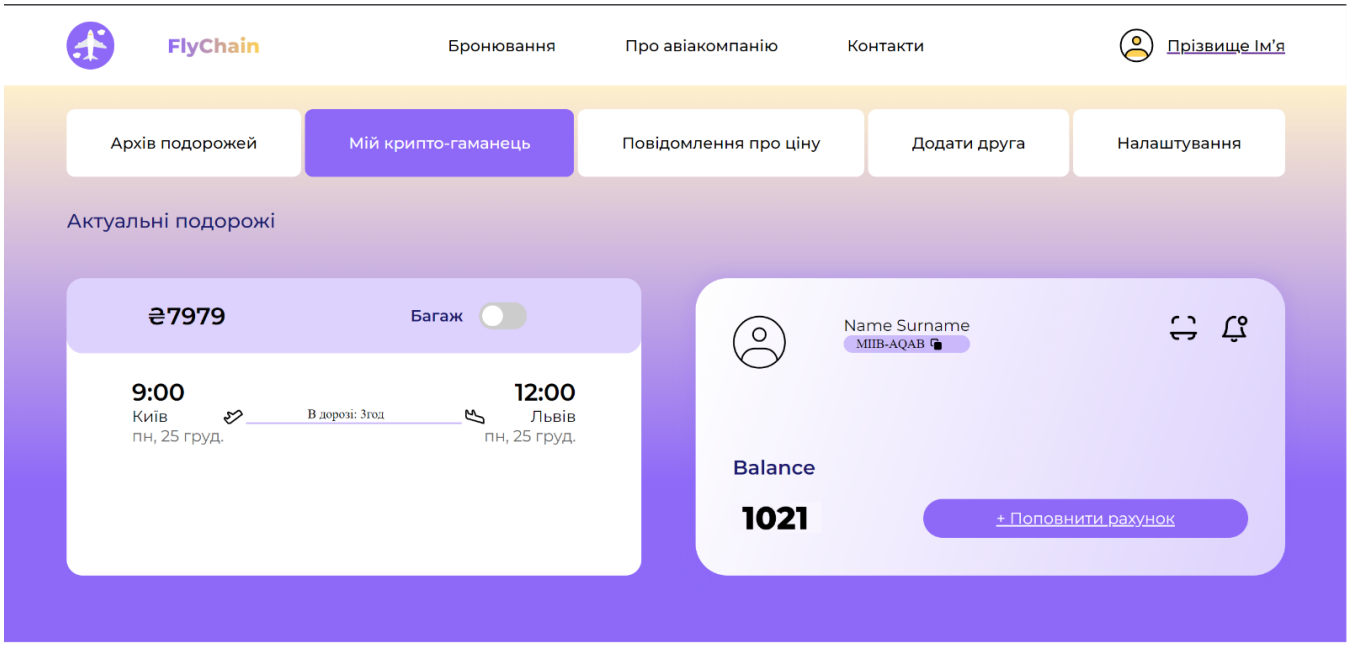

Рис. 4.23. Квиток у особистому кабінеті

4.4. Висновки до розділу

У цьому розділі описано використання вебінтерфейсу для взаємодії користувача з програмною системою. Всі дані надходять безпосередньо з блокчейну. Квитки має додавати компанія в сам блокчейн, проте в подальшому можливо цей процес більш автоматизованим та зручним.

Перша сторінка включає навігаційну панель з кнопками для бронювання, реєстрації, входу та інформації про авіакомпанію та контакти. Користувач може скористатися панеллю пошуку, обравши тип маршруту та вказавши необхідну інформацію.

Реєстрація - це простий процес, де достатньо натиснути одну кнопку, щоб отримати гаманець користувача в блокчейні. Після реєстрації користувач отримує приватний та публічний ключі, які варто зберегти для подальшого входу.

Вхід в особистий кабінет виконується за допомогою збережених під час реєстрації ключів. Особистий кабінет має вкладки з різними функціями, такими як архів подорожей, крипто-гаманець, повідомлення про ціни тощо.

Процес бронювання квитків включає кілька кроків: спочатку вибір маршруту, потім вибір квитка, введення даних пасажира, після чого користувач може обрати, або прибрати пакет послуг, останім кроком є вибір місця. Після завершення бронювання, користувач потрапляє до сторінки з оглядом та підтвердженям даних та робить оплату. В результаті отримує підтвердження та дані про транзакцію. Куплений квиток відображається в особистому кабінеті користувача.

На головній сторінці також розміщено інформацію про популярні напрямки, переваги використання блокчейну та інструкцію щодо створення крипто-гаманця для купівлі авіаквитків.

Отже система в цілому комбінує зручність інтерфейсу для користувача з високою швидкістю блокчейну, створюючи потужний інструмент для придбання авіаквитків.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було розроблено програмну систему для поліпшення процесу придбання авіаквитків з використанням технології блокчейну та вебінтерфейсом для користувачів.

Для досягнення цієї мети було зроблено наступне:

1) розглянуто існуючі проблеми та проведено аналіз різних блокчейн-рішень, які використовуються у сфері сфері купівлі авіаквитків;
2) проаналізовано подібні системи на основі блокчейну, які використовуються для купівлі авіаквитків;
3) обґрунтовано використання бібліотек та інструментів, які використовуються для створення системи;
4) розроблено програмну систему;
5) описано варіанти використання системи.

Ця система представляє собою важливий внесок у розвиток та модернізацію процесу придбання авіаквитків. Її розробка базується на дослідженні сучасних технологій, зокрема технології блокчейну та його інтеграцію в авіакомпанії з вебінтерфейсом для кінцевого користувача. Використання блокчейну забезпечує надійність та недоступність для втручання, що є особливо важливим для авіаційної галузі. В роботі було детально розглянуто переваги використання блокчейну в сфері авіаквитків, вказуючи на можливості контролю та безпеки операцій. Крім того, наведені числові прогнози розвитку ринку блокчейну в авіаційній галузі демонструє його високий потенціал.

Дослідження предметної області надає глибокий аналіз використання технології блокчейн для покращення процесу придбання авіаквитків. Відзначено ключові характеристики блокчейну, такі як децентралізація, незмінність та підвищена безпека, які роблять його ідеальним інструментом для авіаційної промисловості. Також слід відзначити потенціал блокчейну у створенні «розумних квитків», що може забезпечити зручний та безпечний доступ пасажирів до

авіапослуг. В цілому, використання технології блокчейн може призвести до суттєвого покращення ефективності та безпеки авіаційної індустрії.

Технологія блокчейн надає можливість в подальшому створити надійний та неушкоджуваний спосіб підтвердження особистостей, використовуючи біометричні дані. Це допомагає зменшити ризик терористичних актів та підробки документів, що може мати серйозні наслідки для авіаційної безпеки. Крім того, блокчейн дозволяє об'єднати різні сегменти галузі, щоб створити безшовний досвід подорожування для клієнтів. Це не тільки покращує задоволення клієнтів, але й дозволяє економити великі суми на фрагментації та неефективності.

Використання технології блокчейн у сфері купівлі авіаквитків обіцяє значні покращення у безпеці, ефективності та цифровому досвіді подорожування для всіх учасників авіаційної промисловості.

В моїй роботі розглянуто важливість вибору алгоритму консенсусу при впровадженні блокчейну в авіаційній промисловості. В залежності від контексту застосування, вибір алгоритму може вплинути на ефективність та безпеку мережі. Під час розробки системи було враховано проблему з алгоритмом консенсусу *PoW* та вирістано алгоритм *PoS*. Що є обгрунтованим кроком, оскільки вибір правильного алгоритму консенсусу важливий для ефективності та продуктивності блокчейну. Консенсус у блокчейні визначає, як нові дані додаються до попередніх записів. Таким чином, вибір правильного алгоритму консенсусу є ключовим етапом у розгортанні блокчейну в авіаційній промисловості, і він повинен бути узгоджений зі специфічними потребами та контекстом застосування.

У роботі наведено приклади розробок у сфері авіаційної промисловості, де використання блокчейн-технологій може призвести до значних поліпшень. Один з найбільших гравців у цій галузі, компанія *Lufthansa*, активно досліджує можливості впровадження блокчейну. Однією з ключових областей досліджень є використання блокчейну для купівлі авіабілетів. Компанія розробляє прототип блокчейн-платформи, що дозволяє зберігати інформацію про бронювання авіаквитків. Це може значно полегшити та прискорити процес покупки для пасажирів. Крім того, *Lufthansa* досліджує інші потенційні застосування блокчейну у туристичній галузі,

такі як бронювання готелів, прокат автомобілів, страхування подорожей та обмін валют. Ці розробки вказують на потенціал блокчейну для оптимізації та покращення різних аспектів авіаційної промисловості, починаючи від купівлі квитків і закінчуючи додатковими послугами для пасажирів. Загалом в роботі було розглянуто декілька різноманітних інновацій та новаторських рішень в авіаційній галузі, що свідчить про зацікавленість в інтеграції блокчейну до своїх систем від найбільших до наймешних компаній.

Для написання програмного забезпечення був обраний редактор коду *Visual Studio Code*. Його вибір обгрунтовується його потужністю та гнучкістю, що надає розробникам широкий спектр інструментів для написання складних програм. Редактор підтримує різні мови програмування, включаючи *Python*, яка була використана для написання блокчейну та вебінтерфейсу. Його використання дозволило ефективно реалізувати блокчейн-додаток для купівлі авіаквитків. Вибір цих інструментів та мов програмування був обумовлений їхньою потужністю, ефективністю та підтримкою великою спільнотою розробників.

Програмна система включає класи, що реалізовують блокчейн та його функціональність. Головні компоненти включають алгоритм *Proof of Stake* (*PoS*) для створення нових блоків, управління квитками на літаки та забезпечення комунікації між вузлами мережі. У цілому, ця програмна система є комплексним рішенням, яке об'єднує аспекти блокчейну, управління квитками на літаки та мережеву комунікацію між вузлами.

Користувацький інтерфейс програмної системи був розроблений з урахуванням того, щоб користувач отримував задоволення від використання технології блокчейну. Усі дані, які відображаються, надходять безпосередньо з блокчейну. На поточному етапі, компанія додає квитки в блокчейн, але цей процес можна подальшими розробками зробити більш автоматизованим та зручним. Процес реєстрації простий: достатньо натиснути всього одну кнопку, щоб отримати користувацький гаманець у системі. Бронювання квитків включає кілька кроків: вибір маршруту та квитка, введення пасажирських даних, вибір пакету послуг та вибір місця. Отже, вся система об'єднує комфортний користувацький інтерфейс з

високою швидкістю блокчейну, створюючи потужний інструмент для придбання авіаквитків.

В результаті отримано систему, яка використовує передові технології блокчейну з алгоритмом консенсусу *PoS* та авіаквитками всередині кожного блоку. Це дозволяє здійснювати операції швидко (до 3 секунд на даному етапі) та ефективно, без значних витрат на апаратне забезпечення. Схожі системи все ще перебувають на стадіях досліджень та розробки, тимчасом як розроблена мною програмна система вже пропонує надійне та безпечне рішення для придбання авіаквитків за допомогою технології блокчейну. Робота відкриває нові можливості для подальшого розвитку та оптимізації процесів придбання авіаквитків.

Ще однією ключовою особливістю є відсутність комісій завдяки використанню власного блокчейну та токену. Блокчейн дозволяє робити транзакції з будь-якої точки світу. Квитки інтегровані в сам блокчейн, що робить їх більш надійними, ніж при використані смарт-контрактів. Використання кожного квитка у вигляді окремого токена за допомогою унікального *id*, робить їх ще безпечнішими. До того використаний мною алгоритм консенсусу не вимагає великих апаратних ресурсів, не такий шкідливий для природи як деякі, що використовуються в розроблюваних аналогах, а його процес створення блоків можна перетворити в програму лояльності. Блокчейн працює без значних апаратних затрат і без комісій, що згідно досліджень має призвести до зниження вартості квитків. А разом з вебінтерфейсом ще більше спрощує та прискорює процес придбання. Використання технології блокчейну гарантує високий рівень безпеки та надійності транзакцій.

Отже, результати дослідження та розробки свідчать про високий потенціал та переваги використання технології блокчейну для поліпшення процесу придбання авіаквитків. Оскільки подібні системи наразі знаходяться на стадії розробки, розроблена програмна система може мати великий комерційний успіх у випадку вдалого впровадження на ринку авіаквитків. Впровадження цієї системи має потенціал значно полегшити та прискорити процес отримання авіаквитків для кінцевого користувача, а також знизити їх вартість завдяки відсутності комісій.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ПОЛОЖЕННЯ ПРО ДИПЛОМНІ РОБОТИ (ПРОЕКТИ) ВИПУСКНИКІВ НАЦІОНАЛЬНОГО АВІАЦІЙНОГО УНІВЕРСИТЕТУ [Електронний ресурс].- режим доступу: *https://docs.google.com/document/d/106GuMlLn68qMbwUXoSmcsMf9z7iYhI15vFE6CUlF8JI/edit* (дата звернення 25.10.2023). - Назва з екрана.
2. ДОКУМЕНТАЦІЯ. ЗВІТИ У СФЕРІ НАУКИ І ТЕХНІКИ. Структура і правила оформлення. ДСТУ 3008-95 [Електронний ресурс].- режим доступу: *https://drive.google.com/file/d/1yFY-9gxz0a3cuXQvmUpFOzATfdiEFx0K/view* (дата звернення 25.10.2023). - Назва з екрана.
3. Блокчейн для продажи билетов на Олимпиаду-2024 в Париже [Електронний ресурс].- 2022.- режим доступу: *https://ispace.news/blockchain/blokcein-dlya-prodazi-biletov/*(дата звернення 19.10.2023 р). - Назва з екрана.
4. *OPINION: How the Aviation Industry Can Tap Into Blockchain Technology Benefits* [Електронний ресурс].- 2023.- режим доступу: *https://www.aviationtoday.com/2023/04/06/opinion-how-the-aviation-industry-can-tap-into-blockchain-technology-benefits/* (дата звернення 20.10.2023). - Назва з екрана.
5. *Blockchain in Aviation Industry* [Електронний ресурс].- 2022.- режим доступу: *https://www.analyticssteps.com/blogs/blockchain-aviation-industry* (дата звернення 20.10.2023). - Назва з екрана.
6. *How Airlines Can Save Millions with Blockchain* [Електронний ресурс].- 2023.- режим доступу: *https://appinventiv.com/blog/blockchain-in-aviation/#:~:text=How%20is%20blockchain%20used%20in,supply%20chain%20of%20aircraft%20parts.* (дата звернення 20.10.2023). - Назва з екрана.
7. *Future of the Airport - Suitcase Tracking with Blockchain* [Електронний ресурс].- 2018.- режим доступу: *https://www.linkedin.com/pulse/future-airport-suitcase-tracking-blockchain-ceyhun-yakup-%C3%B6zkardes/* (дата звернення 20.10.2023). - Назва з екрана.

8. *Building a blockchain for aviation maintenance records* [Електронний ресурс].- 2021.- режим доступу: *https://www.researchgate.net/publication/349363171_Building_a_blockchain_for_aviation_maintenance_records* (дата звернення 20.10.2023). - Назва з екрана.
9. *Machine Learning Based Predictive Maintenance Model* [Електронний ресурс].- 2022.- режим доступу: *https://www.researchgate.net/publication/366935314_Machine_Learning_Based_Predictive_Maintenance_Model* (дата звернення 20.10.2023). - Назва з екрана.
10. *BLOCKCHAIN IN AVIATION* [Електронний ресурс].- 2018.- режим доступу: *https://www.iata.org/contentassets/2d997082f3c84c7cba001f506edd2c2e/blockchain-in-aviation-white-paper.pdf* (дата звернення 20.10.2023). - Назва з екрана.
11. *What Could Blockchain Do for Airlines?* [Електронний ресурс].- 2019.- режим доступу: *https://www.bcg.com/publications/2019/what-could-blockchain-do-airlines* (дата звернення 20.10.2023). - Назва з екрана.
12. *The Impact of Blockchain Technology in the Aviation Industry* [Електронний ресурс].- 2019.- режим доступу: *https://blog.cryptoflies.com/the-impact-of-blockchain-technology-in-the-aviation-industry/* (дата звернення 20.10.2023). - Назва з екрана.
13. *Blockchain: the future of flight data management?* [Електронний ресурс].- 2018.- режим доступу: *https://www.airport-technology.com/features/blockchain-future-flight-data-management/* (дата звернення 20.10.2023). - Назва з екрана.
14. *Airliners use of blockchain technology* [Електронний ресурс].- 2021.- режим доступу: *https://www.dlapiperintelligence.com/investmentrules/blog/articles/2021/airliner-use-of-blockchain-technology.html* (дата звернення 20.10.2023). - Назва з екрана.
15. Обзор *Air Travel Domain* - часть 4. *Blockchain Challenge* [Електронний ресурс].- 2018.- режим доступу: *https://www.artofba.com/post/overview-air-travel-domain-part4-blockhain-challenge* (дата звернення 20.10.2023). - Назва з екрана.

16. *Applications Of Blockchain In The Aviation Sector* [Електронний ресурс].- режим доступу: *https://prestmit.com/blog/applications-of-blockchain-in-the-aviation-sector/* (дата звернення 20.10.2023). - Назва з екрана.
17. *Malta: The Use Of Blockchain In Aviation* [Електронний ресурс].- 2020.- режим доступу: *https://www.mondaq.com/fin-tech/994262/the-use-of-blockchain-in-aviation* (дата звернення 20.10.2023). - Назва з екрана.
18. *Blockchain Application in Fiji's Aviation Industry* [Електронний ресурс].- 2020.- режим доступу: *https://www.researchgate.net/publication/344943686_Blockchain_Application_in_Fiji's_Aviation_Industry* (дата звернення 20.10.2023). - Назва з екрана.
19. *Blockchain in Aviation | Emerging Applications and Use Cases* [Електронний ресурс].- 2021.- режим доступу: *https://blockchain.oodles.io/blog/blockchain-aviation-applications-use-cases/* (дата звернення 25.10.2023). - Назва з екрана.
20. *How some airlines are getting involved in blockchain, cryptocurrency and NFTs* [Електронний ресурс].- 2022.- режим доступу: *https://thepointsguy.com/news/blockchain-crypto-airlines/* (дата звернення 25.10.2023). - Назва з екрана.
21. *Blockchain for Aviation – Better Transparency and Trust Using Blockchain* [Електронний ресурс].- режим доступу: *https://www.leewayhertz.com/blockchain-aviation-better-transparency-trust/* (дата звернення 25.10.2023). - Назва з екрана.
22. *Globant and TravelX team up to transform the airline industry with blockchain technology* [Електронний ресурс].- 2023.- режим доступу: *https://www.travelx.io/globant-and-travelx-team-up-to-transform-the-airline-industry-with-blockchain-technology/* (дата звернення 25.10.2023). - Назва з екрана.
23. *The future of airline tickets: NFTs are here to stay* [Електронний ресурс].- 2022.- режим доступу: *https://travelx.io/the-future-of-airline-tickets-nfts-are-here-to-stay/* (дата звернення 25.10.2023). - Назва з екрана.

Додаток А

Лістинг коду вебінтерфейсу

Файл *aftercreatewallet.html*

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
                                                                    <link
href="https://fonts.googleapis.com/css2?family=Montserrat:wght@400;500;600;800&dis
play=swap" rel="stylesheet">
    <link rel="stylesheet" href="/static/style.css">
</head>
<body>
    <header class = "header">
        <a href=""><img class = "icon" src="/static/images/Vector.svg" alt="icon"></a>
        <p class="name">FlyChain</p>
        <a href="rclickonreservation" class = "nav__link">Бронювання</a>
        <a href="" class = "nav__link">Про авіакомпанію</a>
        <a href="" class = "nav__link">Контакти</a>

        <a href="" class = "register">Реєстрація</a>
        <button class = "sign__button">Вхід</button>
    </header>

    <main class = "main">
```

```
<section class = "save-info">
  <div class = "form-save-info" >
    <p class = "text-save-info">Дані гаманця</p>
    <p class = "text-save-info">Публічний ключ</p>
    <p class = "login-save-info">{{publicKeyString}}</p>
    <p class = "text-save-info">Приватний ключ</p>
    <p class = "pass-save-info">{{privateKeyString}}</p>

        <button onclick="window.location.href = 'continue_aftersave'" class = "continue-save-info">Продовжити</button>
    </div>

  </section>
</main>
<footer class = "footer">
  <div class = "inside__footer">
    <div class = "flexmenu__footer">
      <div class = "flex-svgtext__footer">
        <div class = "svg__footer">
          <svg xmlns="http://www.w3.org/2000/svg" width="60" height="60" viewBox="0 0 60 60" fill="none">
          <g clip-path="url(#clip0_40_315)">
          <path d="M30 60C46.5685 60 60 46.5685 60 30C60 13.4315 46.5685 0 30 0C13.4315 0 0 13.4315 0
          30C0 46.5685 13.4315 60 30 60Z" fill="#FFD256"/>
            <path d="M45.6732 26.7517C45.5787 29.2911 44.6929 31.2753 43.6063 31.2753C42.5315 31.2753 41.6456
            29.2911 41.5511 26.7517C42.1771 26.5746 42.874 26.4683 43.6063 26.4683C44.3504 26.4683 45.0472 26.
            5746 45.6732 26.7517Z" fill="white"/>
```

<path d="M18.4488 26.7517C18.3543 29.2911 17.4685 31.2753 16.3819 31.2753C15.3071 31.2753 14.4213 29.2911 14.3268 26.7517C14.9528 26.5746 15.6496 26.4683 16.3819 26.4683C17.126 26.4683 17.8347 26.5746 18.4488 26.7517Z" fill="white"/>

<path d="M30.8503 18.8857L31.181 31.2991L50.0433 34.0511V31.4763L30.8503 18.8857Z" fill="#E6E9EE"/>

<path d="M29.1496 18.8857L28.8189 31.2991L9.95667 34.0511V31.4763L29.1496 18.8857Z" fill="#E6E9EE"/>

<path d="M30.4015 40.5942L30.555 46.4171L39.3897 47.7045V46.4997L30.4015 40.5942Z" fill="#E6E9EE"/>

<path d="M29.5984 40.5942L29.4449 46.4171L20.6102 47.7045V46.4997L29.5984 40.5942Z" fill="#E6E9EE"/>

<path d="M34.004 22.0394C34.004 33.0591 32.2087 49.9488 30 49.9488C27.7914 49.9488 25.9961 33.0591 25.9961 22.0394C25.9961 11.0197 30 10.063 30 10.063C30 10.063 34.004 11.0197 34.004 22.0394Z" fill="white"/>

<path d="M32.7638 17.4093C31.9842 17.0314 31.0275 16.807 30 16.807C28.9724 16.807 28.0157 17.0314 27.2362 17.3975C27.6378 16.1101 28.7244 15.2007 30 15.2007C31.2756 15.2007 32.3622 16.1338 32.7638 17.4093Z" fill="#8E69F8"/>

<path d="M48.1417 13.3821C48.6023 12.3309 48.1299 11.1026 47.0787 10.6419C46.5 10.3821 45.8622 10.4175 45.3307 10.6774C45.1299 10.4057 44.8464 10.1931 44.5275 10.0396C43.4764 9.57894 42.248 10.0514 41.7874 11.1026C41.7638 11.1616 41.7401 11.2207 41.7165 11.2797C41.6456 11.2325 41.563 11.1852 41.4803 11.1498C40.4291 10.6892 39.2008 11.1616 38.7401 12.2128C38.2795 13.264 38.7519 14.4923 39.8031

14.953C39.8976 15.0002 40.0039 15.0356 40.1102 15.0593C40.2165 15.7443 40.6535 16.3585 41.3267

16.6656C42.1181 17.0081 43.0039 16.8309 43.5945 16.2758C43.7716 16.4648 43.996 16.6183 44.2441

16.7246C45.2953 17.1852 46.5236 16.7128 46.9842 15.6616C47.1614 15.26 47.1968 14.823 47.126 14.4215C47.5512

14.2325 47.9291 13.8663 48.1417 13.3821Z" fill="white"/>

<path d="M17.4331 42.2835C17.8937 41.2323 17.4213 40.0039 16.3701 39.5433C15.7914 39.2835 15.1536 39.3189

14.6221 39.5787C14.4213 39.3071 14.1378 39.0945 13.8189 38.9409C12.7677 38.4803 11.5394 38.9528 11.0788

40.0039C11.0551 40.063 11.0315 40.122 11.0079 40.1811C10.937 40.1339 10.8544 40.0866 10.7717

40.0512C9.72049 39.5905 8.49215 40.063 8.03152 41.1142C7.57089 42.1653 8.04333 43.3937 9.09451

43.8543C9.189 43.9016 9.2953 43.937 9.4016 43.9606C9.5079 44.6457 9.9449 45.2598 10.6181 45.5669C11.4095

45.9094 12.2953 45.7323 12.8858 45.1772C13.063 45.3661 13.2874 45.5197 13.5355 45.626C14.5866 46.0866 15.815

45.6142 16.2756 44.563C16.4528 44.1614 16.4882 43.7244 16.4173 43.3228C16.8425 43.1339 17.2205 42.7677

17.4331 42.2835Z" fill="white"/>

</g>

<defs>

<clipPath id="clip0_40_315">

<rect width="60" height="60" fill="white"/>

</clipPath>

</defs>

</svg>

* FlyChain *

```
            </div>
                <p class = "text-undersvg">Один світ, одна валюта</p>
        </div>
        <nav class = "menu__footer">
            <p class = "title-menuok__footer">Меню</p>
            <a href="" class = "item-menu__footer">Бронювання</a>
            <a href="" class = "item-menu__footer">Про компанію</a>
            <a href="" class = "item-menu__footer">Контакти</a>
        </nav>
        <nav class = "ok__footer">
            <p class = "title-menuok__footer">Особистий кабінет</p>
            <a href="" class = "item-ok__footer">Мої бронювання</a>
            <a href="" class = "item-ok__footer">Баланс гаманця</a>
            <a href="" class = "item-ok__footer">Історія замовлень</a>
        </nav>
    </div>
    <div class = "svg-link__footer">
            <svg xmlns="http://www.w3.org/2000/svg" width="35" height="34" viewBox="0 0 35 34" fill="none">
            <path d="M29.5208 4.97927C26.3099 1.76827 22.0408 0 17.5 0C12.9592 0 8.69008 1.76827 5.47921
            4.97927C2.26827 8.19008 0.5 12.4592 0.5 17C0.5 21.5408 2.26827 25.8099 5.47921 29.0208C8.69008
            32.2317 12.9592 34 17.5 34C22.0408 34 26.3099 32.2317 29.5208 29.0207C32.7317 25.8099 34.5 21.5408
            34.5 17C34.5 12.4592 32.7317 8.19008 29.5208 4.97927ZM17.5 32.9263C8.71817 32.9263 1.57366 25.7818
            1.57366 17C1.57366 8.21817 8.71817 1.07366 17.5 1.07366C26.2818 1.07366 33.4263 8.21817 33.4263 17C33.4263
            25.7818 26.2818 32.9263 17.5 32.9263Z" fill="white" fill-opacity="0.8"/>
```

<path d="M12.4894 13.6H8.91048C8.61398 13.6 8.37366 13.8403 8.37366 14.1368V24.8737C8.37366 25.1702 8.61405 25.4105 8.91048 25.4105H12.4894C12.786 25.4105 13.0263 25.1702 13.0263 24.8737V14.1368C13.0263 13.8403 12.786 13.6 12.4894 13.6ZM11.9526 24.3368H9.44738V14.6737H11.9526V24.3368Z" fill="white" fill-opacity="0.8"/>

<path d="M10.3421 7.15786C8.86194 7.15786 7.65786 8.36194 7.65786 9.84207C7.65786 11.3222 8.86194 12.5263 10.3421 12.5263C11.8222 12.5263 13.0263 11.3222 13.0263 9.84207C13.0263 8.36194 11.8222 7.15786 10.3421 7.15786ZM10.3421 11.4526C9.45402 11.4526 8.73152 10.7301 8.73152 9.84207C8.73152 8.95409 9.45395 8.23152 10.3421 8.23152C11.2301 8.23152 11.9526 8.95402 11.9526 9.84207C11.9526 10.7301 11.2302 11.4526 10.3421 11.4526Z" fill="white" fill-opacity="0.8"/>

<path d="M22.5106 13.6C21.4445 13.6 20.3942 13.876 19.4684 14.39V14.1369C19.4684 13.8404 19.228 13.6001 18.9316 13.6001H16.0684C15.7719 13.6001 15.5316 13.8405 15.5316 14.1369V24.8737C15.5316 25.1702 15.772 25.4106 16.0684 25.4106H19.6474C19.9439 25.4106 20.1842 25.1702 20.1842 24.8737V21.0195C20.1842 19.7573 20.4636 18.2527 21.7948 18.2527C22.7881 18.2527 23.1956 19.0907 23.3385 20.0405C23.3777 20.301 23.6034 20.4926 23.8669 20.4926C24.1975 20.4926 24.4461 20.1962 24.3951 19.8696C24.1256 18.1448 23.2112 17.179 21.7948 17.179C20.0889 17.179 19.1106 18.5789 19.1106 21.0195V24.3369H16.6053V14.6737H18.3948V15.3796C18.3948 15.546 18.4641 15.7087 18.5942 15.8125C18.7963 15.9736 19.0746 15.968 19.2675 15.8133C20.1972 15.0678 21.3186 14.6737 22.5106 14.6737C25.6009

14.6737 26.9842 17.28 26.9842 19.8631V24.3368H24.479V22.569C24.479 22.2734 24.2393 22.0337 23.9437

22.0337H23.9406C23.645 22.0337 23.4054 22.2734 23.4054 22.569V24.8736C23.4054 25.1701 23.6458 25.4104 23.9422

25.4104H27.5212C27.8177 25.4104 28.058 25.17 28.058 24.8736V19.863C28.0579 16.1755 25.7767 13.6 22.5106 13.6Z"

fill="white" fill-opacity="0.8"/>

</svg>

<svg xmlns="http://www.w3.org/2000/svg" width="34" height="34" viewBox="0 0 34 34" fill="none">

<path d="M28.6672 4.9792C25.5508 1.76826 21.4072 0 17 0C12.6185 0 8.43088 1.78712 5.33269 4.9792C2.21625 8.19007

0.5 12.4592 0.5 17C0.5 20.5398 1.54588 23.9327 3.52452 26.8119C5.4594 29.6274 8.13459 31.7473 11.2608 32.9424C11.4708

33.0227 11.7136 32.9572 11.8581 32.7629C11.9297 32.6665 11.9632 32.5455 11.9632 32.4241V22.7263C11.9632 22.4298

11.7298 22.1895 11.4421 22.1895H7.10005V18.2527H11.4421C11.7299 18.2527 11.9632 18.0123 11.9632 17.7158V17C11.9632

12.329 16.2647 8.23157 21.1684 8.23157H23.4263V12.1684H21.1684C19.698 12.1684 18.3609 12.6045 17.4035 13.3962C16.3442

14.2722 15.7842 15.5184 15.7842 17V17.7158C15.7842 18.0123 16.0175 18.2526 16.3052 18.2526H18.3595C18.6473 18.2526

18.8805 18.0122 18.8805 17.7158C18.8805 17.4193 18.6472 17.1789 18.3595 17.1789H16.8263V17C16.8263 14.2263 19.1653

13.2421 21.1683 13.2421H23.9473C24.2351 13.2421 24.4683 13.0017 24.4683 12.7053V7.69474C24.4683 7.39824 24.235 7.15792

23.9473 7.15792H21.1683C18.6213 7.15792 16.0396 8.20182 14.0853 10.0219C12.0448 11.9224 10.921 14.4006 10.921

17V17.179H6.57888C6.2911 17.179 6.05785 17.4194 6.05785 17.7158V22.7264C6.05785 23.0229 6.29117 23.2632 6.57888

23.2632H10.921V31.6468C5.28119 29.1573 1.54208 23.3674 1.54208 17C1.54208 8.21815 8.47644 1.07365 16.9998

1.07365C25.514 1.07365 32.4578 8.22785 32.4578 17C32.4578 25.7818 25.5234 32.9262 16.9999 32.9262C16.9429 32.9262

16.885 32.9259 16.8262 32.9251V23.2631H23.9473C24.2351 23.2631 24.4683 23.0227 24.4683 22.7263V17.7158C24.4683

17.4193 24.235 17.1789 23.9473 17.1789H20.8976C20.6098 17.1789 20.3765 17.4193 20.3765 17.7158C20.3765 18.0123 20.6098

18.2526 20.8976 18.2526H23.4263V22.1894H16.3052C16.0174 22.1894 15.7842 22.4298 15.7842 22.7262V33.4466C15.7842

33.7338 16.0038 33.9706 16.2823 33.983C16.5405 33.9944 16.7753 33.9999 17 33.9999C21.4073 33.9999 25.5508 32.2317

28.6672 29.0207C31.7654 25.8286 33.5 21.5142 33.5 16.9998C33.4999 12.4592 31.7836 8.19007 28.6672 4.9792Z"

fill="white" fill-opacity="0.8"/>

</svg>

<svg xmlns="http://www.w3.org/2000/svg" width="35" height="34" viewBox="0 0 35 34" fill="none">

<path d="M29.5208 4.97921C26.3099 1.76827 22.0408 0 17.5 0C12.9592 0 8.69008 1.76827 5.47921 4.97921C2.26827

8.19008 0.5 12.4592 0.5 17C0.5 21.5408 2.26827 25.8099 5.47921 29.0208C8.69008 32.2317 12.9592 34 17.5 34C22.0408

34 26.3099 32.2317 29.5208 29.0208C32.7317 25.8099 34.5 21.5408 34.5 17C34.5 12.4592 32.7317 8.19008 29.5208

4.97921ZM17.5 32.9263C8.71817 32.9263 1.57366 25.7818 1.57366 17C1.57366 8.21817 8.71817 1.07366 17.5 1.07366C26.2818

1.07366 33.4263 8.21817 33.4263 17C33.4263 25.7818 26.2818 32.9263 17.5 32.9263Z" fill="white" fill-opacity="0.8"/>

<path d="M29.4765 10.7838C29.3149 10.6441 29.0808 10.6202 28.8871 10.7101C28.6094 10.8389 28.3255 10.9521 28.0367 11.0492C28.3232 10.6585 28.5519 10.2243 28.7133 9.7588C28.7798 9.56682 28.7467 9.34867 28.6091 9.19919C28.4369 9.01233 28.1627 8.97301 27.9475 9.0956C27.1629 9.54271 26.3167 9.84639 25.4286 9.99992C24.4353 9.08823 23.1575 8.58945 21.8 8.58945C18.9902 8.58945 16.677 10.7571 16.4445 13.5078C15.864 13.4172 14.7283 13.0934 14.4567 13.0046C12.4924 12.3356 10.7228 11.1576 9.33847 9.5975C9.2564 9.50499 9.14828 9.4362 9.0267 9.41395C8.78232 9.36932 8.5505 9.49271 8.44597 9.70209C8.10192 10.3916 7.92741 11.133 7.92741 11.9058C7.92741 12.9745 8.26196 13.9817 8.8646 14.8097C8.6768 14.7745 8.47692 14.8404 8.34583 15.0016C8.27046 15.0943 8.23274 15.212 8.22829 15.3314C8.2263 15.3859 8.22451 15.4405 8.22451 15.4956C8.22451 16.9444 8.88791 18.2611 9.9601 19.1206C9.88606 19.1707 9.82364 19.2399 9.78061 19.3252C9.71666 19.4517 9.70955 19.6003 9.75179 19.7355C10.2201 21.2335 11.3835 22.3744 12.8259 22.842C11.5195 23.6373 10.0083 24.0662 8.46437 24.0662C8.32804 24.0662 8.18772 24.0626 8.0474 24.0558C7.77872 24.0435 7.53773 24.231 7.49152 24.5036C7.4538 24.726 7.56868 24.9484 7.76292 25.063C9.73638 26.227 11.9953 26.8421 14.2966 26.8421C17.0433 26.8421 19.6595 25.9868 21.8408 24.4034C22.1099 24.208 22.1396 23.8176 21.9045 23.5825C21.7151 23.3931 21.417 23.3725 21.2001 23.5296C19.256 24.9373 16.8699 25.7684 14.2966 25.7684C12.8543 25.7684 11.4303 25.5049 10.096 24.9974C11.7099 24.714 13.2354 24.0141 14.506 22.958C14.6595 22.8303 14.7404 22.6285 14.7017 22.4326C14.6523 22.1825 14.4379 22.0056 14.1899 21.9987C12.8826 21.9616 11.7156 21.2393 11.0889 20.1353C11.3878

20.121 11.6869 20.0801 11.9814 20.0131C12.206 19.9619 12.3877 19.7825 12.4169 19.554C12.4521 19.2789 12.2752

19.0277 12.0129 18.9632C10.6106 18.6179 9.57926 17.4781 9.3473 16.0907C9.75836 16.1871 10.1856 16.2296 10.6121

16.2166C10.8753 16.2087 11.098 16.0101 11.1296 15.7407C11.1544 15.5285 11.0417 15.323 10.8593 15.2117C9.69527

14.5024 9.00093 13.2679 9.00093 11.9058C9.00093 11.5729 9.0423 11.2473 9.12438 10.932C10.5317 12.3235 12.2398

13.3838 14.1106 14.0211C14.1277 14.0269 15.9761 14.6024 16.6762 14.6053C16.7219 14.6064 16.9402 14.616 16.9405

14.616C17.1655 14.6251 17.3837 14.4886 17.4687 14.2623C17.4931 14.1974 17.502 14.1275 17.5003 14.0582C17.4996 14.0269

17.4988 13.9956 17.4988 13.9642C17.4988 11.5926 19.4282 9.66317 21.7999 9.66317C22.9611 9.66317 24.0496 10.1195

24.865 10.948C24.9831 11.068 25.1494 11.1271 25.3161 11.104C25.8625 11.0282 26.3956 10.9031 26.9124 10.73C26.6373

11.0317 26.313 11.2903 25.9493 11.4939C25.7042 11.6312 25.6002 11.9294 25.7157 12.1855L25.7236 12.2031C25.8182

12.4124 26.0351 12.5381 26.2639 12.5165C26.708 12.4741 27.1459 12.401 27.5756 12.298C27.1954 12.69 26.7774 13.0462

26.3239 13.3639C26.176 13.4675 26.0909 13.6392 26.0971 13.8197L26.0982 13.8537C26.0995 13.8904 26.1009 13.9272

26.1009 13.9642V14.0127C26.0885 17.0203 24.9413 19.7671 23.0682 21.8483C22.8768 22.0608 22.8888 22.3873 23.0911

22.5895C23.311 22.8095 23.6701 22.7988 23.8783 22.5675C25.9828 20.2295 27.1471 17.243 27.1743 14.074C28.132 13.3606

28.9403 12.4885 29.58 11.478C29.7207 11.2559 29.6812 10.9608 29.4765 10.7838Z" fill="white" fill-opacity="0.8"/>

</svg>

* *

```
        <p class = "company__footer">ⓒ FlyChain 2023</p>
      </div>
    </footer>
</body>
</html>
```

Файл *complete_tx.html*

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
                                                                    <link
href="https://fonts.googleapis.com/css2?family=Montserrat:wght@400;500;600;800&dis
play=swap" rel="stylesheet">
    <link rel="stylesheet" href="/static/style.css">
</head>
<body>
    <header class = "header">
        <a href=""><img class = "icon" src="/static/images/Vector.svg" alt="icon"></a>
        <p class="name">FlyChain</p>
        <a href="clickonreservation" class = "nav__link">Бронювання</a>
        <a href="" class = "nav__link">Про авіакомпанію</a>
        <a href="" class = "nav__link">Контакти</a>

          <a href="" class = "inacc__header"><svg xmlns="http://www.w3.org/2000/svg"
width="48" height="48" viewBox="0 0 48 48" fill="none">
```

```
<path d="M0 0H48V48H0V0Z" fill="white" fill-opacity="0.01"/>
<path d="M10 38C10.3443 32.974 14.6872 29 19.9942 29H28.0071C33.3071 29 37.6456 32.9635 38 37.98" fill="#FFD256"/>
<path d="M10 38C10.3443 32.974 14.6872 29 19.9942 29H28.0071C33.3071 29 37.6456 32.9635 38 37.98" stroke="black" stroke-width="2" stroke-linecap="round" stroke-linejoin="round"/>
<path fill-rule="evenodd" clip-rule="evenodd" d="M24.0001 44C35.0458 44 44.0001 35.0457 44.0001 24C44.0001 12.9543 35.0458 4 24.0001 4C12.9544 4 4.00006 12.9543 4.00006 24C4.00006 35.0457 12.9544 44 24.0001 44Z" stroke="black" stroke-width="2" stroke-linecap="round" stroke-linejoin="round"/>
<path d="M24.0001 23.0001C26.7615 23.0001 29.0001 20.7615 29.0001 18.0001C29.0001 15.2387 26.7615 13.0001 24.0001 13.0001C21.2387 13.0001 19.0001 15.2387 19.0001 18.0001C19.0001 20.7615 21.2387 23.0001 24.0001 23.0001Z" fill="#FFD256" stroke="black" stroke-width="2" stroke-linejoin="round"/>
</svg> <span>Прізвище Ім’я</span></a>
</header>
<main class = "main">
<section class = "complete-tx">
<div class = "svg-complete-tx">
<svg xmlns="http://www.w3.org/2000/svg" width="52" height="38" viewBox="0 0 52 38" fill="none">
<path d="M49 3L17.375 35L3 20.4545" stroke="#02C816" stroke-width="6" stroke-linecap="round" stroke-linejoin="round"/>
</svg>
</div>
<h2 class = "complete-tx__h2">Дякуємо за замовлення!</h2>
<h3 class = "complete-tx__h3">Листа з квитанцією надіслано на вашу пошту</h3>
<div class = "form-complete-tx">
```

```
                <p class = "title-form-complete-tx">Дякуємо. Ваше замовлення отримано!</p>
            <div class = "top-form-complete-tx">
                <div class = "detail-top-form-complete-tx">
                    <p class = "staticdetail-top-form-complete-tx">ID транзакції:</p>
                                <p class = "dymanicdetail-top-form-complete-tx">1c4b049b667811ee9fbc088fc34164a0</p>
                </div>
                <div class = "horizontal-purplerec"></div>
                <div class = "detail-top-form-complete-tx">
                    <p class = "staticdetail-top-form-complete-tx">Дата:</p>
                    <p class = "dymanicdetail-top-form-complete-tx">20 жовт, 2023</p>
                </div>
                <div class = "horizontal-purplerec"></div>
                <div class = "detail-top-form-complete-tx">
                    <p class = "staticdetail-top-form-complete-tx">Сума:</p>
                    <p class = "dymanicdetail-top-form-complete-tx">₴7979</p>
                </div>
            </div>
            <div class = "bottom-form-complete-tx">
                <p class = "title-bottom-form-complete-tx">Деталі замовлення</p>
                <div class = "productsum-bottom-form-complete-tx">
                    <p class = "product-bottom-form-complete-tx">Продукт</p>
                    <p class = "sum-bottom-form-complete-tx">Сума</p>
                </div>
                <div class = "purplerec"></div>
                <div class = "flex-items-bottom-form-complete-tx">
                        <p class = "item-product-bottom-form-complete-tx">Лондон — Париж:</p>
                    <p class = "item-sum-bottom-form-complete-tx">₴589.00</p>
```

```
            </div>
            <div class = "purplerec"></div>
            <div class = "flex-items-bottom-form-complete-tx">
               <p class = "item-product-bottom-form-complete-tx">Пакет послуг:</p>
               <p class = "item-sum-bottom-form-complete-tx">₴250.00</p>
            </div>
            <div class = "purplerec"></div>
            <div class = "flex-items-bottom-form-complete-tx">
               <p class = "fullsum-bottom-form-complete-tx">Всього:</p>
               <p class = "item-fullsum-bottom-form-complete-tx">₴7979</p>
            </div>
         </div>
      </div>
   </section>
</main>
</body>
</html>
```