МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА КРИПТОГАМАНЦЯ З ВИКОРИСТАННЯМ PYTHON І ETHEREUM»**

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
М.О. Волковінський
(ініціали та прізвище)

Керівник професор кафедри програмної інженерії, професор, д.т.н. Чопоров С.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Волковінському Михайлу Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка криптогаманця з використанням Python і Ethereum

керівник роботи Чопоров Сергій Вікторович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка криптогаманця.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	01.02.2024	
3.	Обробка методичних та теоретичних	15.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	19.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2024	

Студент _____ М.О. Волковінський
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.В. Чопоров
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка криптогаманця з використанням Python і Ethereum»: 38 с., 20 рис., 1 табл., 8 джерел.

КРИПТОГАМАНЕЦЬ, CSS, DJANGO, ETHEREUM, HTML, POSTGRESQL, PYTHON.

Об’єкт дослідження – розробка криптогаманця з використанням Python та Ethereum.

Предмет дослідження – Python, Ethereum, їх взаємодія та бібліотека Django для роботи з серверною частиною додатку.

Мета роботи: розробити криптогаманець використовуючи Python та Ethereum.

Методи дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проєктування, конструювання та тестування програмного забезпечення.

При розробці додатку був проведений аналіз предметної області, обрано та спроєктовано архітектуру системи. Реалізовано серверну частину з використанням PYTHON, а саме використовуючи бібліотеку DJANGO, клієнтська частина була розроблена за допомогою TailwindCSS.

В результаті роботи отримано криптогаманець.

SUMMARY

Bachelor’s qualifying paper “Development of a Crypto Wallet Using Python and Ethereum”: 38 pages, 20 figures, 1 table, 8 references.

CRYPTO WALLET, CSS, DJANGO, ETHEREUM, HTML, POSTGRESQL, PYTHON.

The object of the study is the development of a crypto wallet using Python and Ethereum.

The subject of the study is Python, Ethereum, their interaction and Django library for server-side work of the application.

The aim of the study is to build a crypto wallet using Python and Ethereum.

The methods of research are methods of collecting and analyzing software requirements, methods of modelling, designing, constructing, and testing software.

During the application development process the subject domain, utilized software frameworks and libraries were analyzed, the database structure and system architecture were designed. The server part was implemented using Python, namely its tools such as Django framework. The client side was developed using the TailwindCSS.

The result is a crypto wallet.

ЗМІСТ

ВСТУП

Цей дипломний проєкт присвячений розробці криптогаманця з використанням мови програмування Python та технології Ethereum. Вибір цих інструментів зумовлений їх високою безпекою та простотою застосування, що робить їх ідеальними для створення криптовалютних додатків.

Мета проєкту полягає в розробці безпечного та функціонального криптогаманця. Для досягнення цієї мети буде виконано кілька ключових завдань: дослідження можливостей Python і Ethereum, розробка серверної частини додатку з використанням бібліотеки Django та створення клієнтської частини за допомогою TailwindCSS.

Особлива увага приділяється проєктуванню системи. Крім того, буде створено зручний інтерфейс користувача для забезпечення легкого та ефективного використання гаманця.

Ця робота складається з трьох основних частин, кожна з яких охоплює різні аспекти розробки криптогаманця.

У першому розділі досліджено та сформульовано вимоги до розробки застосунку. Тут детально описано критерії, які має задовольняти криптогаманець, включаючи функціональні та нефункціональні вимоги. Також проведено аналіз доступних технологій та інструментів, і на основі цього аналізу обрано найбільш підходящі для реалізації проєкту. Серед обраного інструментарію – мова програмування Python, платформа Ethereum для забезпечення безпеки транзакцій, та бібліотека Django для розробки серверної частини додатку.

У другому розділі зосереджено увагу на проєктуванні застосунку та бази даних. Тут розглянуто архітектурні рішення, які забезпечують надійну роботу криптогаманця, включаючи його основні компоненти та взаємодію між ними.

У третьому розділі наведено конкретні зразки реалізації застосунку. Цей пункт містить описи коду, застосованого для розробки як серверної

інфраструктури, так і клієнтського інтерфейсу криптогаманця. Для серверної частини була використана мова програмування Python та фреймворк Django, а для забезпечення зручного та інтуїтивно прозорого інтерфейсу – бібліотека TailwindCSS.

1 РОБОТА З PYTHON ТА DJANGO

1.1 Мова програмування Python

Python – потужна і популярна мова програмування, яку широко використовують для розроблення додатків і вебсайтів. Одна з ключових переваг Python – наявність великої екосистеми фреймворків, що надають розробникам готові інструменти та рішення для прискорення процесу розробки. Поговоримо про це докладніше [7].

Python нескладно освоїти, по ньому є величезна кількість посібників, як платних, так і безкоштовних. При цьому Python – дуже перспективна мова програмування. Вивчивши його, не буде сумнівів, що через пару років не доведеться переходити на щось нове.

Python працює з фреймворками Django, Pyramid, Pylons і Flask. З Django – найбільшою мірою. Цей фреймворк простий, безпечний і швидкий, відрізняється високою надійністю і стабільністю в роботі. Тож не дивно, що розробники зазвичай віддають перевагу саме йому.

Одна з головних переваг Python криється в самій його філософії – читабельності коду.

Синтаксис Python банально простіше зрозуміти. Тут навіть немає фігурних дужок, як в інших мовах програмування. І з таких дрібниць складається вся мова Python.

Прості і зрозумілі інструменти налагодження. Python Debugger (PDB) – потужний відладчик, простий у використанні і відмінно задокументований. Навіть новачкам нескладно буде з ним розібратися.

Python можна використовувати далеко не тільки в веброзробці. Його застосовують в машинному навчанні, обробці зображень, розробці десктопних і мобільних додатків, NLP і так далі [1].

Розширення сфери використання Python сприяло і стрімкому зростанню ком’юніті. У Мережі величезна кількість матеріалів по Python, є відмінні курси навчання, а на будь-які питання нескладно знайти відповіді на тематичних форумах або в Python Software Foundation.

1.2 Аналіз вибору фреймворка

Django – вебфреймворк, написаний на Python. Вебфреймворк – програмне забезпечення, яке підтримує розробку динамічних вебсайтів, застосунків і служб. Він надає набір інструментів і функціональних можливостей, які вирішують багато загальних проблем, пов’язаних із розробкою вебсайтів: безпека, доступ до БД, сесії, обробка шаблонів, маршрутизація URL-адрес, інтернаціоналізація, локалізація тощо [2].

Використання фреймворків, таких як Django, дозволяє швидко розробляти безпечні й надійні вебзастосунки стандартизованим способом, без необхідності повторно винаходити колесо.

Що такого особливого в Django? Це фреймворк Python, що означає, що для цього фреймворку вже розробили велику кількість бібліотек з відкритим вихідним кодом. Якщо потрібно вирішити конкретну проблему, є вірогідність, що хтось вже реалізував для неї бібліотеку.

Django – один із найпопулярніших вебфреймворків, написаних на Python. Він безумовно найповніший і пропонує широкий спектр нестандартних функцій, таких як: автономний вебсервер для розробки та тестування, кешування, система проміжного програмного забезпечення, ORM, механізм шаблонів, обробка форм та інтерфейс юніт-тестування Python. Django також постачається з «батарейками в комплекті» (battery included), пропонуючи вбудовані застосунки, такі як система автентифікації, адміністративний інтерфейс з автоматично створеними сторінками для операцій CRUD, створення каналів синдикації (RSS / Atom), мап сайтів. Існує

навіть фреймворк геоінформаційної системи (ГІС), побудований в Django. У таблиці 1.1 проілюстровано усі аспекти кожного з фреймворків, та обгрунтування вибору саме Django.

Таблиця 1.1 – Порівняння фреймворків

Параметр	**Django**	**Flask**	**FastAPI**
Тип фреймворку	Full-stack	Micro	Micro
Випуск	2005	2010	2018
Призначення	Побудова комплексних вебдодатків	Легкі вебдодатки	Високопродуктивні API
Архітектура	MVT (Model-View-Template)	WSGI (Web Server Gateway Interface)	ASGI (Asynchronous Server Gateway Interface)
Головні переваги	Повний набір вбудованих інструментів	Простота та гнучкість	Висока продуктивність та асинхронність
Вбудована ORM	Так (Django ORM)	Ні	Ні
Роутінг	Вбудований	Ручний	Вбудований, з підтримкою типів
Швидкодія	Помірна	Висока	Дуже висока
Асинхронна підтримка	Обмежена	Через додаткові модулі	Вбудована
Документація	Вбудована адміні панель та багата документація	Проста та зрозуміла	Автоматична документація (Swagger)

Продовження таблиці 1.1

Параметр	**Django**	**Flask**	**FastAPI**
Підтримка REST	Через Django REST framework	Через Flask-RESTful та інші бібліотеки	Нативна підтримка
Модулі та розширення	Великий набір сторонніх пакетів	Широкий вибір сторонніх модулів	Підтримка сторонніх бібліотек
Громада	Велика та активна	Велика та активна	Зростаюча
Крива навчання	Стрімка через велику кількість функцій	Полога через простоту	Помірна через нові концепції

Під час вибору фреймворку для проєкту ключовими факторами на користь Django стали його повноцінний набір вбудованих інструментів, які забезпечують швидку і ефективну розробку складних вебдодатків. На відміну від Flask і FastAPI, які є мікрофреймворками, Django пропонує повний стек, що включає в себе все необхідне для створення вебдодатків з нуля. Це дозволяє зосередитися на бізнес-логіці додатку, не витрачаючи багато часу на налаштування основних компонентів. Django включає вбудовану ORM, що значно спрощує роботу з базами даних і дозволяє швидко створювати складні моделі даних.

Однією з ключових переваг Django є його багата документація та велика спільнота розробників. Це означає, що у випадку виникнення будь-яких проблем або питань під час розробки, завжди можна знайти підтримку та ресурси, які допоможуть їх вирішити. Крім того, вбудована адмін панель Django значно спрощує управління додатком та роботу з даними, що є важливим аспектом при розробці криптогаманця, де безпека та надійність є критичними факторами.

Зрештою, хоча Django може поступатися FastAPI у швидкості та асинхронній підтримці, його надійність та зрілість як фреймворку роблять його ідеальним вибором для розробки масштабованих і безпечних додатків. Django дозволяє легко інтегрувати додаткові функції та модулі, що забезпечує гнучкість у розширенні можливостей криптогаманця в майбутньому. Використання Django також спрощує дотримання найкращих практик у розробці вебдодатків, що сприяє створенню якісного та надійного продукту.

Для початку використання Django необхідно встановити його на комп`ютері та створити проєкт. Процес встановлення Django наведено нижче [3].

Встановлення pip. Найлегше використовувати автономний інсталятор pip. Якщо pip вже встановлений, можливо, доведеться оновити його, якщо він застарілий. Якщо він застарілий, установка не працюватиме.

Інструмент venv надає ізольовані середовища Python, які є більш практичними, ніж встановлення пакетів на всю систему. Він також дозволяє встановлювати пакети без прав адміністратора. У підручнику з внесення змін описується, як створити віртуальне середовище.

Після того, як було створено та активовано віртуальне середовище, потрібно ввести команду з рисунка 1.1.

```
$ python -m pip install Django
```

Рисунок 1.1 – Команда встановлення Django

Далі потрібно створити проєкт та додаток відповідно офіційної інструкції вписав у командний рядок наступні команди (див. рис. 1.2).

```
$ django-admin startproject mysite
$ python manage.py runserver
```

Рисунок 1.2 – Створення проєкту та запуск сервера

На рисунку 1.3 зображено очікуваний результат при запуску сервера.

```
C:\Users\drake\PG\Projects\Diploma Project\crypto_wallet>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
June 11, 2024 - 19:03:20
Django version 5.0.1, using settings 'crypto_wallet.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Рисунок 1.3 – Результат при успішному налаштуванні та запуску сервера

Переконавшись, що сервер працює як заплановано, потрібно створити перший додаток в нашому проєкті за допомогою команди (див. рис. 1.4).

```
$ python manage.py startapp app_name polls index.")
```

Рисунок 1.4 – Команда створення додатку в проєкті

Далі потрібно написати логіку додатку у файлі views.py. На рисунку 1.5 проілюстрований зразок логіки додатку Polls (опитування) у polls/views.py.

```
from django.http import HttpResponse



def index(request):
    return HttpResponse("Hello, world. You're at the polls index.")
```

Рисунок 1.5 – Зразок логіки додатку

Далі потрібно підключити цю логіку до самого додатку як зображено на рисунку 1.6.

```
from django.urls import path

from . import views

urlpatterns = [
    path("", views.index, name="index"),
]
```

Рисунок 1.6 – Підключення логіки до додатку

Далі потрібно підключити логіку додатку до логіки проєкту як зображено на рисунку 1.7.

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path("polls/", include("polls.urls")),
    path("admin/", admin.site.urls),
]
```

Рисунок 1.7 – Підключення логіки додатку до логіки проєкту

На данному прикладі було розглянуто як створити, налаштувати проєкт та додаток в Django, де і як реалізовувати логіку додатку та як підключити її до проєкта. Стандартною практикою вважається створювати окремий додаток для кожної частини функціоналу, щоб забезпечити модульність і зручність у керуванні кодом. Наприклад, створення окремого додатку для роботи з користувачами, реєстрацією та аутентифікацією дозволяє легко підтримувати,

тестувати та розширювати цей функціонал незалежно від інших частин проєкту. Це сприяє кращій організації коду, полегшує повторне використання модулів у різних проєктах та дозволяє іншим розробникам швидко орієнтуватися у структурі проєкту, розуміючи, що кожен додаток відповідає за конкретну функціональність.

1.3 Висновки до розділу 1

Розробка криптогаманця з використанням Django є важливим та відповідальним процесом, який потребує глибокого розуміння сучасних технологій та стандартів безпеки.

Django, як високорівневий вебфреймворк на Python, забезпечує надійний фундамент для створення захищених та масштабованих вебдодатків. Його вбудовані механізми безпеки, такі як захист від CSRF та XSS атак, а також можливість легкої інтеграції з різними базами даних, роблять його ідеальним вибором для розробки фінансових додатків. Крім того, використання окремих додатків для різних аспектів функціональності, таких як управління користувачами, обробка транзакцій та забезпечення безпеки, дозволяє підтримувати високий рівень організованості коду та спрощує процеси розширення та підтримки системи.

Використання Django для створення криптогаманця не лише сприяє розробці надійного та ефективного рішення, але й забезпечує гнучкість та можливість подальшого масштабування проєкту відповідно до потреб користувачів.

2 АНАЛІЗ ТА ФОРМУЛЮВАННЯ ВИМОГ КРИПТОГАМАНЦЯ

2.1 Вимоги до проєкту

Для дослідження предметної області з використанням Python, Ethereum та Django було обрано тему розробки криптовалютного гаманця. Проєкт передбачає створення вебдодатку, який дозволяє користувачам керувати своїми криптовалютними активами, зокрема Ethereum, здійснювати транзакції та переглядати історію операцій. Основна мета проєкту – створити безпечний та зручний вебдодаток для управління криптовалютними активами. Проєкт передбачає розробку функцій, таких як створення та керування криптовалютними гаманцями, можливість відправляти та отримувати криптовалюту, а також перегляд історії транзакцій. Розробка цих функцій дозволить зробити проєкт максимально зручним та привабливим для користувачів, що дозволить привернути більше уваги до проєкту та збільшити його популярність.

Основні функціональні вимоги до проєкту [4]:

- підтримка відправлення та отримання Ethereum транзакцій;
- перегляд історії транзакцій з можливістю фільтрації за датою та сумою;
- інтеграція з Ethereum блокчейном для забезпечення актуальності даних;
- підтримка мобільних пристроїв та різних браузерів для зручного використання гаманця;
- доступність через HTTPS-протокол для забезпечення безпеки взаємодії користувачів;
- зручний та зрозумілий інтерфейс.

2.2 Використані інструменти та середовище розробки

PyCharm – це інтегроване середовище розробки (IDE) для Python, розроблене компанією JetBrains. PyCharm забезпечує розширені можливості для редагування коду, зокрема підсвічування синтаксису, автодоповнення, налагодження та інтеграцію з системами контролю версій. IDE підтримує платформи Windows, macOS та Linux, що робить його зручним для використання на різних операційних системах.

Переваги PyCharm:

- багатофункціональність – PyCharm має вбудовані інструменти для налагодження, тестування, роботи з базами даних та управління версіями;
- підтримка веброзробки: IDE підтримує популярні фреймворки, такі як Django, Flask, та інші;
- інтеграція з системами контролю версій: PyCharm підтримує Git, Mercurial та інші системи контролю версій;
- розширюваність – можна встановлювати додаткові плагіни для розширення функціональності IDE;
- зручний інтерфейс – PyCharm має інтуїтивно зрозумілий інтерфейс, що дозволяє зосередитися на розробці програмного забезпечення.

Узагальнюючи, PyCharm – це потужне інтегроване середовище розробки, яке надає широкий набір інструментів для ефективної розробки Python-додатків.

Python – це інтерпретована, високорівнева мова програмування загального призначення. Вона широко використовується для веброзробки, аналізу даних, штучного інтелекту, автоматизації задач та багато іншого. Python має простий синтаксис, який робить його легким для навчання та використання, а також велику кількість бібліотек та фреймворків, які спрощують розробку додатків.

2.3 Використані бібліотеки та фреймворки

Django – це високорівневий фреймворк для веброзробки на мові Python, що сприяє швидкому розвитку та практичному дизайну. Він включає в себе потужні утиліти для розробки, такі як ORM (Object-Relational Mapping), система маршрутизації URL, шаблони для формування HTML-сторінок та багато іншого. Django допомагає розробникам створювати безпечні та підтримувані вебдодатки.

Основні переваги Django:

- **висока швидкість розробки**: Django дозволяє швидко створювати додатки завдяки використанню готових компонентів;
- **безпека**: фреймворк має вбудовані захисти від багатьох поширених вебзагроз, таких як SQL-ін’єкції, XSS-атаки, CSRF-захист та інші;
- **масштабованість**: Django дозволяє розробляти додатки, які можуть обробляти великі обсяги даних і велике навантаження;
- **велика спільнота**: Django має активну спільноту розробників, які створюють та підтримують велику кількість плагінів та розширень.

2.4 Основні функціональні можливості

У даному розділі визначені основні функціональні особистості розробленого застосунку.

Прецедент «Реєстрація».

Функція: реєстрація особистого профілю у системі.

Основний потік подій: на екрані реєстрації користувач вводить необхідні дані для створення акаунту, і натискає кнопку «Sign Up». При успішній реєстрації виконується зберігання даних користувача (Нікнейм , пошту, ім’я користувача, фамілія користувача, пароль, фотографія профіля

зберігається у відповідну директорію) до бази даних PosgreSQL.

Альтернативний хід подій: якщо користувач з введеними даними вже зареєстрований, буде відображена помилка з відповідним повідомленням.

Виключення 1: якщо введено некоректні дані, буде відображена помилка з відповідним повідомленням.

Виключення 2: якщо одно з обов'язкових рядків не було заповнено, то буде відображена помилка з відповідним повідомленням, та відповідний рядок буде виділений.

Прецедент «Вхід до системи».

Функція: вхід до особистого профілю у системі.

Основний потік подій: користувач вводить коректні дані та натискає кнопку «Sign In». Після чого відбувається вхід до відповідного профіля, якщо процес авторизації пройшов успішно.

Альтернативний хід подій: якщо користувач з введеними даними не зареєстрований у системі, буде відображена помилка з відповідним повідомленням.

Виключення 1: якщо введено некоректні дані, буде відображена помилка з відповідним повідомленням.

Прецедент «Редагування профілю».

Функція: редагування особистого профіля.

Основний потік подій: користувач натискає на кнопку «Settings», після чого редагує необхідні дані профіля, після чого натискає на кнопку «Save». Нові дані зберігаються в системі, а користувача перенаправлено на сторінку профіля.

Альтернативний потік подій: якщо ніяких змін внесено не було, після натискання на кнопку «Save» користувача буде перенаправлено на сторінку профіля.

Передумова: користувач повинен бути зареєстрованим та увійти до системи.

Виключення 1: при введенні некоректних даних буде відображена помилка з відповідним повідомленням та буде запропоновано здійснити ще одну спробу змінити дані.

Прецедент «Вихід із системи».

Функція: користувач виходить з особистого профіля.

Основний потік подій: користувач натискає кнопку «Sign out» на навігаційній панелі. При успішному виконанні операції, користувача буде виведено з акаунту системи та перенаправлено на головну сторінку.

Передумова: користувач повинен увійти до системи.

Прецедент «Пошук за гаманцем».

Функція: користувач здійснює пошук транзакцій за адресою криптогаманця.

Основний потік подій: користувач вводить необхідну адресу криптогаманця в поле пошуку та натискає кнопку «Search». Якщо криптогаманець з такою адресою існує, то користувачу буде відображені останні 20 транзакцій за даним гаманцем.

Альтернативний хід подій: якщо гаманець з введеною адресою не існує, користувачу буде відображено відповідне повідомлення.

Прецедент «Додати гаманець до улюблених».

Функція: користувач додає криптогаманець до улюблених.

Основний потік подій: користувач вводить адресу криптогаманця в поле пошуку, натискає на чекбокс з назвою «Favorite» та натискає кнопку «Search». Після чого в особистому профілі буде доданий криптогаманець за відповідною адресою.

Альтернативний хід подій: якщо гаманець з введеною адресою не існує, користувачу буде відображено відповідне повідомлення, а в особистому профілі не буде додано жодного запису.

На рисунку 2.1 за допомогою діаграми використання проілюстровані можливості взаємодії користувача з системою.

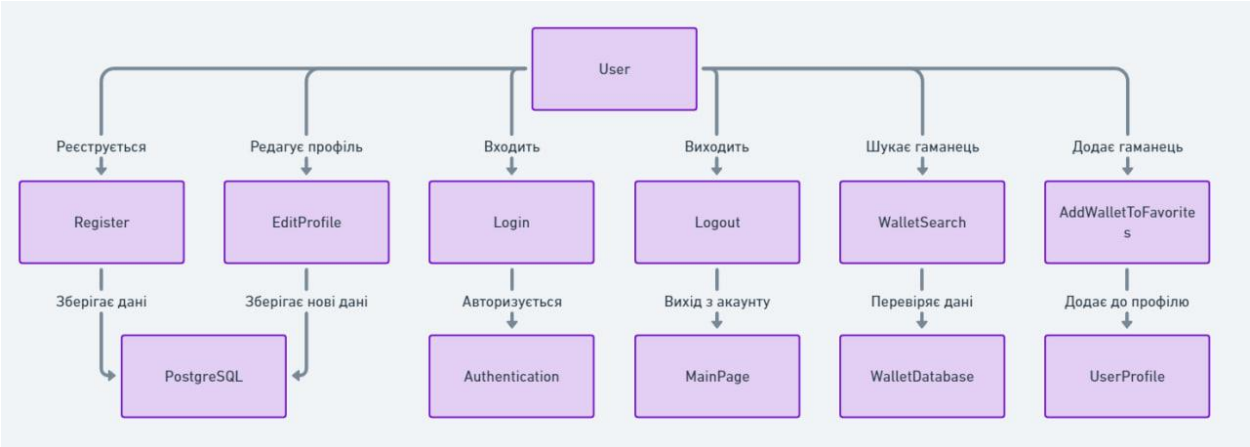

Рисунок 2.1 – Діаграма використання

2.5 Проєктування бази даних

PostgreSQL це система керування базами даних корпоративного класу з відкритим кодом. Вона підтримує як формат SQL, так і JSON для реляційних і нереляційних запитів для розширюваності та відповідності SQL. PostgreSQL підтримує розширені типи даних і функції оптимізації продуктивності, які доступні лише в дорогих комерційних базах даних, наприклад Oracle і SQL Server. Він також відомий як Postgres [8].

Нижче наведено основні переваги/переваги PostgreSQL:

– PostgreSQL може запускати динамічні вебсайти та вебпрограми як варіант стека LAMP;
– PostgreSQL журналування з випереджальним записом робить її високостійкою до відмов;
– PostgreSQL вихідний код є у вільному доступі за ліцензією з відкритим кодом – це дає вам свободу використовувати, змінювати та впроваджувати його відповідно до потреб вашого бізнесу;
– PostgreSQL підтримує географічні об’єкти, тому його можна виокристовувати для служб на основі визначення місцезнаходження та систем географічної інформації;
– PostgreSQL підтримує географічні об’єкти, тому його можна

використовувати як сховище геопросторових даних для служб на основі визначення місцезнаходження та систем географічної інформації;

– щоб вивчити Postgres, вам не потрібно багато навчання, оскільки він простий у використанні;
– низькі витрати на технічне обслуговування та адміністрування для вбудованих і корпоративних систем PostgreSQL.

Django ORM (Object-Relational Mapping) є потужним інструментом, який дозволяє вам працювати з базами даних за допомогою об’єктів Python, уникаючи написання сирого SQL-коду. ORM відповідає за трансформацію моделей Django у таблиці баз даних. Опишемо як це відбувається крок за кроком.

Визначення моделі – потрібно визначити клас моделі у файлі models.py. Кожен клас моделі унаочнює таблицю бази даних, а кожен атрибут класу представляє стовпець таблиці (див. рис. 2.2).

```
class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    name = models.CharField(max_length=200, null=True, blank=True)
    email = models.EmailField(max_length=50, null=True, blank=True)
    last_name = models.CharField(max_length=200, null=True, blank=True)
    short_intro = models.CharField(max_length=200, null=True, blank=True)
    account_photo = models.ImageField(null=True, blank=True, upload_to='profiles/', default='profiles/default.png')
    created = models.DateTimeField(auto_now_add=True)
    id = models.UUIDField(default=uuid.uuid4, unique=True, primary_key=True, editable=False)

    def __str__(self):
return str(self.user.username)
```

Рисунок 2.2 – Модель користувача в Джанго

Створення міграцій – Django використовує міграції для відстеження змін у моделях і відображення цих змін у базі даних. Команда makemigrations генерує міграційні файли на основі визначених моделей: python manage.py makemigrations.

Застосування міграцій – команда migrate застосовує міграції до бази даних, створюючи або оновлюючи таблиці відповідно до визначених моделей: python manage.py migrate.

Django аналізує файли міграцій і генерує відповідні SQL-запити для створення або зміни таблиць.

Таким чином вищезазначена модель Джанго перетворюється в SQL запит до бази даних PostgreSQL (див. рис. 2.3).

```
CREATE TABLE profile (
    user_id INTEGER NOT NULL PRIMARY KEY REFERENCES auth_user(id),
    name VARCHAR(200) NULL,
    email VARCHAR(50) NULL,
    last_name VARCHAR(200) NULL,
    short_intro VARCHAR(200) NULL,
    account_photo VARCHAR(100) NULL DEFAULT 'profiles/default.png',
    created TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP,
    id UUID NOT NULL PRIMARY KEY DEFAULT uuid_generate_v4()
);
```

Рисунок 2.3 – Аналог моделі Джанго у вигляді SQL запиту

2.6 Висновки до розділу 2

У другому розділі було проведено аналіз та формулювання вимог до криптовалютного гаманця, а також розглянуто використані інструменти та середовище розробки. Основна мета проєкту полягала у створенні безпечного

та зручного вебдодатку для управління криптовалютними активами, зокрема Ethereum, що включає підтримку відправлення та отримання криптовалюти, перегляд історії транзакцій, інтеграцію з блокчейном Ethereum, підтримку мобільних пристроїв та різних браузерів, а також доступність через HTTPS-протокол.

Для розробки проєкту було використано інтегроване середовище розробки PyCharm, яке забезпечує розширені можливості для редагування коду, налагодження, тестування, роботи з базами даних та управління версіями. Python, як основна мова програмування, відзначається своєю простотою у навчанні та використанні, а також великою кількістю бібліотек та фреймворків, що спрощують розробку додатків.

Основним фреймворком для розробки вебдодатку було обрано Django, який надає потужні інструменти для швидкого розвитку та практичного дизайну, включаючи ORM, систему маршрутизації URL, шаблони для формування HTML-сторінок та вбудовані захисти від поширених вебзагроз. Django забезпечує високу швидкість розробки, безпеку та масштабованість, що дозволяє створювати додатки, здатні обробляти великі обсяги даних і навантаження.

Також у розділі було розглянуто основні функціональні можливості розробленого застосунку, такі як реєстрація та вхід до системи, редагування профілю, вихід з системи, пошук за адресою гаманця та додавання гаманця до улюблених. Проєктування бази даних здійснювалося з використанням PostgreSQL, що забезпечує високу стійкість до відмов та підтримку складних типів даних, а Django ORM дозволяє працювати з базами даних за допомогою об’єктів Python, уникаючи написання сирого SQL-коду.

Загалом, другий розділ підкреслює важливість детального аналізу вимог та використання сучасних інструментів і технологій для створення надійного та функціонального криптовалютного гаманця. Це забезпечує високу якість кінцевого продукту та зручність його використання для користувачів.

3 ПРОГРАМНА ЧАСТИНА ДОДАТКУ

3.1 Пошук транзакцій за допомогою API

У цьому розділі буде детально розглянуто програмну реалізацію пошуку транзакцій за криптовалютним гаманцем, що було здійснено шляхом інтеграції з API вебсайту Etherscan. Розробка цієї функціональності дозволяє користувачам отримувати інформацію про всі транзакції, пов’язані з вказаною адресою гаманця, включаючи деталі про відправника, отримувача, обсяги переказів, витрати на газ та поточний баланс. Вся інформація збирається у реальному часі, що забезпечує актуальність даних та надає користувачам повну картину їх криптовалютних операцій.

Використання Etherscan API для взаємодії з блокчейном Ethereum дозволяє швидко і ефективно отримувати необхідні дані, уникаючи складностей безпосередньої роботи з блокчейном. Розроблена система включає налаштування середовища розробки, інтеграцію з API, обробку запитів та представлення отриманих даних у зручному форматі для користувача. Даний підхід забезпечує високу надійність та безпеку, що є критичним для управління криптовалютними активами [5].

У наведеному фрагменті коду реалізується звернення до API Etherscan (див. рис. 3.1).

```
def make_api_url(module, action, address, **kwargs):
    API_KEY = '1WYR1RVJPVI96EMSU3YCB43AH5MMYVU95E'
    BASE_URL = 'https://api.etherscan.io/api'
    url = BASE_URL +
f"?module={module}&action={action}&address={address}&apikey={API_KEY}"
    for key, value in kwargs.items():
        url += f"&{key}={value}"
    return url
```

Рисунок 3.1 – Звернення до API Etherscan

В даному фрагменті коду реалізована функція додавання гаманця за адресою до списку улюблених в особистий профіль користувача. Спочатку перевіряємо чи гаманець з даною адресою є в списку улюблених в особистому профілі і тільки потім додається до списку улюблених, якщо до цього моменту його там не було (див. рис. 3.2).

```
def search_address(request):
    form = AddressForm()
    context = {'form': form}
    if request.method == 'POST':
        form = AddressForm(request.POST)
        if form.is_valid():
            address = form.cleaned_data['address']
            transactions = get_transactions(address)
            context['address'] = address
            context['transactions'] = transactions
            if request.user.is_authenticated and 'save_address' in request.POST:
                profile = Profile.objects.get(user=request.user)
                # Check if the user already has 10 addresses
                if profile.favorite_addresses.count() < 10:
                    favorite_address, created = FavoriteAddress.objects.get_or_create(profile=profile,
address=address)
                    if created:
                        messages.success(request, 'Address saved to your profile.')
                    else:
                        messages.info(request, 'Address already in your favorites.')
                else:
                    messages.error(request, 'You can only save up to 10 favorite addresses.')
    return render(request, 'wallet/search_address.html', context)
```

Рисунок 3.2 – Додавання гаманця до «улюблених»

В наступному фрагменті проілюстрована реалізація пошуку транзакцій за адресою криптогаманця, що ввів користувач у поле для пошуку. Процес відбувається наступним чином – спочатку потрібно зробити http запит до API вебресурсу Etherscan, з отриманої відповіді треба вилучити потрібні дані такі як «кому», «адресат», «сума переказу», «витрачений газ», «поточний баланс», «час транзакції», а потім передаємо ці дані через змінну «context» на фронт-енд та відображаємо їх на клієнтській стороні (див. рис. 3.3).

```
def get_transactions(address):
    ETHER_VALUE = 10 ** 18

    transactions_url = make_api_url("account", "txlist", address, startblock=0,
endblock=99999999, page=1, offset=10, sort="asc")
    response = get(transactions_url)
    data = response.json()["result"]
    internal_tx_url = make_api_url("account", "txlistinternal", address, startblock=0,
endblock=99999999, page=1, offset=10, sort="asc")
    response2 = get(internal_tx_url)
    data2 = response2.json()["result"]
    data.extend(data2)
    data.sort(key=lambda x: int(x["timeStamp"]))
    current_balance = 0
    context = {}
    trans_num = 0

    for tx in data:
        to = tx["to"]
        from_address = tx["from"]
        value = int(tx["value"]) / ETHER_VALUE

        if "gasPrice" in tx:
            gas = int(tx["gasUsed"]) * int(tx['gasPrice']) / ETHER_VALUE
        else:
            gas = int(tx["gasUsed"]) / ETHER_VALUE
        time = datetime.fromtimestamp(int(tx["timeStamp"]))
        money_in = to.lower() == address.lower()

        if money_in:
            current_balance += value
        else:
            current_balance -= value + gas

        context[trans_num] = {
            "To": to,
            "From": from_address,
            "Value": value,
            "Gas_Spent": gas,
            "Current_Balance": current_balance,
            "Time": time
        }

        trans_num += 1
    return context
```

Рисунок 3.3 – Пошук транзакцій

В останньому фрагменті коду для цього розділу розглянемо функцію, що забезпечує коректну роботу головної сторінки та сторінки пошуку (див. рис. 3.4).

```
def main_page_view(request):
    context = {}
    return render(request, 'wallet/homepage.html', context)
```

Рисунок 3.4 – Логіка відображення головної сторінки

3.2 Робота з користувачами

В першу чергу розглянемо фрагмент коду з реєстрацією користувача.

Функція register починається з перевірки, чи метод запиту – POST. Якщо так, то створюється форма користувача CustomUserCreationForm з даними POST.

Далі, копіюються дані POST і видаляються поля, які не потрібні для профілю. Після цього ініціалізується форма профілю ProfileForm з даними та файлами POST. Якщо обидві форми валідні, починається атомарна транзакція.

У транзакції зберігається користувач у базі даних, перевіряється наявність профілю для користувача, оновлюються та зберігаються дані профілю.

Після цього виконується авторизація щойно створеного користувача, відображається повідомлення про успішну реєстрацію, і відбувається перенаправлення на головну сторінку.

Якщо виникає помилка цілісності, виводиться відповідне повідомлення. Якщо метод запиту – GET, ініціалізуються пусті форми CustomUserCreationForm та ProfileForm, після чого рендериться сторінка реєстрації з цими формами (див. рис. 3.5).

```
def register(request):
    if request.method == 'POST':
        user_form = CustomUserCreationForm(request.POST)

        profile_data = request.POST.copy()
        for key in ['username', 'email', 'password1', 'password2', 'csrfmiddlewaretoken']:
            profile_data.pop(key, None)

        profile_form = ProfileForm(profile_data, request.FILES)

        if user_form.is_valid() and profile_form.is_valid():
            try:
                with transaction.atomic():
                    user = user_form.save()

                    # Check if a profile already exists
                    profile, created = Profile.objects.get_or_create(user=user)
                    profile.name = profile_form.cleaned_data.get('name')
                    profile.last_name = profile_form.cleaned_data.get('last_name')
                    profile.short_intro = profile_form.cleaned_data.get('short_intro')
                    if 'account_photo' in request.FILES:
                        profile.account_photo = request.FILES['account_photo']
                    profile.save()

                    login(request, user)
                    messages.success(request, 'Registration successful. Welcome to our site!')
                    return redirect('home')
            except IntegrityError as e:
                messages.error(request, 'There was an error creating your profile. Please try again.')
        else:
            messages.error(request, 'Registration failed. Please correct the errors below.')
    else:
        user_form = CustomUserCreationForm()
        profile_form = ProfileForm()

    return render(request, 'registration/register.html', {
        'user_form': user_form,
        'profile_form': profile_form,
    })
```

Рисунок 3.5 – Реєстрація користувача

Функція loginUser починається зі створення форми LoginForm з даними POST, якщо вони є. Якщо метод запиту – POST і форма валідна, з форми отримуються очищені дані для імені користувача та пароля. Потім виконується автентифікація користувача за допомогою authenticate.

Якщо користувач успішно автентифікований, виконується вхід у систему за допомогою функції login, і відбувається перенаправлення на головну сторінку.

Якщо автентифікація не вдається, виводиться повідомлення про неправильні облікові дані. Незалежно від методу запиту, наприкінці функції рендериться сторінка входу з формою (див рис. 3.6).

```
class LoginForm(forms.Form):
    username = forms.CharField()
    password = forms.CharField(widget=forms.PasswordInput)

def loginUser(request):
    form = LoginForm(request.POST or None)
    if request.method == 'POST' and form.is_valid():
        username = form.cleaned_data['username']
        password = form.cleaned_data['password']

        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect('home')
        else:

            messages.error(request, 'Invalid login credentials. Please try again.')
    return render(request, 'registration/login.html', {'form': form})
```

Рисунок 3.6 – Логін користувача

Тепер розглянемо фунцію для розлогування з профіля. Функція logout_view починається з виклику функції logout, яка виходить із поточного сеансу користувача. Після цього відображається повідомлення про успішний вихід. Нарешті, виконується перенаправлення на головну сторінку (див. рис. 3.7).

```
def logout_view(request):
    logout(request)
    messages.success(request, 'You logged out successfully')
    return redirect('home')
```

Рисунок 3.7 – Вихід з профіля користувача

В наступному фрагменті коду розглянемо функцію, що відповідає за оновлення даних профіля. Функція update_profile починається з перевірки автентифікації користувача за допомогою декоратора @login_required. Далі, отримується профіль користувача за допомогою get_object_or_404. Якщо метод запиту – POST, ініціалізується форма ProfileUpdateForm з даними POST і файлами, прив’язаними до поточного профілю.

Якщо форма валідна, збереження форми оновлює профіль, після чого виводиться повідомлення про успішне оновлення і відбувається перенаправлення на сторінку профілю. Якщо метод запиту – GET, ініціалізується форма з поточними даними профілю. Наприкінці функції рендериться сторінка налаштувань з формою (див. рис. 3.8).

На даному фрагменті коду проілюстрована функція, що відповідає за відображення сторінки особистого профілю користувача. Функція profile починається з перевірки автентифікації користувача за допомогою декоратора @login_required. Далі, отримується профіль користувача за допомогою get_object_or_404, використовуючи поточного користувача. Потім отримуються всі улюблені адреси, пов’язані з цим профілем, за допомогою profile.favorite_addresses.all().

Наприкінці функції рендериться сторінка профілю з передачею об’єктів профілю та улюблених адрес у контексті (див. рис. 3.9).

```
@login_required
def update_profile(request):
    profile = get_object_or_404(Profile, user=request.user)
    if request.method == 'POST':
        form = ProfileUpdateForm(request.POST, request.FILES, instance=profile)
        if form.is_valid():
            form.save()
            messages.success(request, 'Your profile was successfully updated!')
            return redirect('profile')
    else:
        form = ProfileUpdateForm(instance=profile)
    return render(request, 'registration/settings.html', {'form': form})
```

Рисунок 3.8 – Оновлення інформації профіля користувача

```
@login_required
def profile(request):
    profile = get_object_or_404(Profile, user=request.user)
    favorite_addresses = profile.favorite_addresses.all()
    return render(request, 'profile.html', {'profile': profile, 'favorite_addresses':
favorite_addresses})
```

Рисунок 3.9 – Відображення сторінки профіля корситувача

І остання функція, яку розглянемо відповідає за видалення «улюблених» криптогаманців з особистого профіля користувача. Функція delete_favorite_address починається з перевірки автентифікації користувача за допомогою декоратора @login_required. Далі, отримується профіль користувача за допомогою get_object_or_404, використовуючи поточного

користувача. Потім отримується улюблена адреса за допомогою get_object_or_404, використовуючи address_id та профіль. Після цього адреса видаляється за допомогою методу delete(). Виводиться повідомлення про успішне видалення улюбленої адреси. Нарешті, виконується перенаправлення на сторінку профілю за допомогою HttpResponseRedirect і функції reverse (див. рис. 3.10).

```
@login_required
def delete_favorite_address(request, address_id):
    profile = get_object_or_404(Profile, user=request.user)
    address = get_object_or_404(FavoriteAddress, id=address_id, profile=profile)
    address.delete()
    messages.success(request, 'Favorite address was successfully deleted.')
    return HttpResponseRedirect(reverse('profile'))
```

Рисунок 3.10 – Видалення гаманця з «улюблених»

3.2 Висновки до розділу 3

Третій розділ зосереджений на програмній розробці криптовалютного гаманця за допомогою Python та Ethereum. Завдяки інтеграції з Etherscan API було реалізовано надійну функціональність пошуку транзакцій, що дозволяє користувачам отримувати деталі транзакцій у реальному часі, такі як відправник, отримувач, обсяги переказів, витрати «газу» та поточний баланс. Це забезпечує актуальність та повноту інформації про криптовалютні операції користувачів [6].

Крім того, було продемонстровано реалізацію різних функцій управління користувачами, включаючи реєстрацію користувачів, вхід, вихід та оновлення профілю, що були розроблені для забезпечення безпечного та зручного користувацького досвіду. Надані фрагменти коду детально

пояснюють, як обробляються дані користувачів та як безпечно обробляються та відображаються транзакції.

Також було розглянуто систему додавання та управління улюбленими адресами гаманців у профілях користувачів. Ця функція дозволяє користувачам легко відстежувати та керувати найважливішими криптовалютними адресами, що покращує зручність та функціональність гаманця.

У цілому, цей розділ підкреслює успішну інтеграцію кількох технологій та фреймворків для створення безпечного, ефективного та зручного криптовалютного гаманця. Детальний аналіз та кроки реалізації забезпечують високу якість розробленої системи, що робить її надійним інструментом для управління криптовалютними активами.

ВИСНОВКИ

Кваліфікаційна робота бакалавра на тему «Розробка криптогаманця з використанням Python та Ethereum» спрямована на створення безпечного та функціонального криптогаманця, використовуючи сучасні технології та фреймворки. Основними інструмен тами для реалізації проєкту стали мова програмування Python, платформа Ethereum та вебфреймворк Django.

Процес розробки складався з кількох ключових етапів.

Вивчення та вибір технологій: аналіз можливостей Python та фреймворку Django для створення серверної частини додатку. Було проведено порівняння з іншими фреймворками, такими як Flask та FastAPI, з урахуванням їх переваг і недоліків.

Розробка серверної частини: створення моделей для зберігання даних криптогаманця у базі даних PostgreSQL, налаштування представлень для обробки запитів користувачів та інтеграція з блокчейном Ethereum за допомогою бібліотеки web3.py. Використання Django ORM значно спростило роботу з базами даних та забезпечило ефективне управління даними.

Розробка клієнтської частини: використання TailwindCSS для створення зручного та інтуїтивно зрозумілого інтерфейсу користувача. Було розроблено HTML-шаблони для відображення даних про гаманці та транзакції, а також налаштовано форми для введення даних користувачами.

Тестування та налагодження: проведення тестових транзакцій для перевірки коректності роботи гаманця, використання інструментів PyCharm для налагодження коду та оптимізації його ефективності.

У процесі роботи було реалізовано основні функціональні можливості криптогаманця, такі як реєстрація та вхід користувачів, редагування профілю, пошук за адресою гаманця, відправлення та отримання криптовалюти, перегляд історії транзакцій, а також додавання адрес до улюблених.

Завдяки використанню сучасних інструментів та методів розробки

вдалося створити надійний та зручний у використанні криптогаманець, який відповідає всім поставленим вимогам. Цей проєкт демонструє можливості інтеграції різних технологій для створення комплексних вебдодатків, що мають високу безпеку та продуктивність.

Кваліфікаційна робота показує важливість детального проєктування та належного тестування програмного забезпечення для досягнення високої якості кінцевого продукту. Використання Django та інших інструментів значно спрощує процес розробки, дозволяючи зосередитися на реалізації бізнес-логіки та задоволенні потреб користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Phillips D. Python 3 Object-oriented Programming: Building robust and maintainable software with object oriented design patterns in Python, 2nd Edition. Packt Publishing, 2015. 460 p.
2. George N. Mastering Django: Core. Packt Publishing, 2016. 694 p.
3. Django. *Django Project*. URL: https://www.djangoproject.com/ (дата звернення 10.04.2024).
4. How to set up a crypto wallet. Coinbase. URL: https://www.coinbase.com/en-ca/learn/tips-and-tutorials/how-to-set-up-a-crypto-wallet (дата звернення: 05.04.2024).
5. Wood G. Ethereum: A Secure Decentralized Generalised Transaction Ledger. Ethereum Project Yellow Paper. 2014. T. 151.
6. Kok A. S. Hands-On Blockchain for Python Developers: Gain blockchain programming skills to build decentralized applications using Python. Packt Publishing, 2019. 450 p.
7. Introduction to Python frameworks: what they are and what they are used for. Foxminded. URL: https://foxminded.ua/freimvorky-python/ (дата звернення: 19.04.2024).
8. What is PostgreSQL? Introduction, advantages and disadvantages. GURU99. URL: https://www.guru99.com/uk/introduction-postgresql.html#features-of-postgresql (дата звернення: 30.04.2024).