Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Сідорова Дмитра Валерійовича*
(ПІБ)

академічної групи *122-21ск-1*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка веб-застосунку для організації обміну криптовалют з використанням Django.*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Спірінцев В.В.*			
розділів:				
спеціальний	*доц. Спірінцев В.В.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексеєв
(підпис) (прізвище, ініціали)

« » 2024 Року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-21ск-1* *Сідорова Дмитра Валерійовича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка веб-застосунку для організації обміну криптовалют з використанням Django.*

затверджена наказом ректора НТУ «ДП» від *23.04.2024* № *375-С*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***20.05.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***3.06.2024 р.***

Завдання видав _____ *доц. Спірінцев В.В*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Сідоров І.С*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: *04.06.2024 р.*

РЕФЕРАТ

Пояснювальна записка: 90 с., 20 рис., 1 табл., 3 дод., 20 джерел.

Об'єкт розробки: веб-орієнтований застосунок для обміну криптовалютами.

Мета кваліфікаційної роботи: розробка веб-орієнтованого застосунку для обміну криптовалют, що забезпечує зручність та безпеку обмінних операцій, з використанням сучасних технологій та інструментів веб-розробки.

У вступі розглядається аналіз та стан проблеми, конкретизується мета кваліфікаційної роботи та сфера її застосування, наводиться обґрунтування актуальності теми.

У першому розділі проаналізовано предметну область, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовано існуючі рішення, обрано платформу для розробки, виконано проектування та розробку веб-орієнтованої програми. Описано роботу програми, алгоритм та структуру її функціонування, а також виклик та завантаження програми, визначено вхідні та вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, здійснено підрахунок вартості роботи зі створення додатка та розраховано час його створення.

Практичне завдання полягає у створенні веб-орієнтованої програми, що забезпечує зручну та безпечну платформу для обміну криптовалютами, спрощує процес обміну та сприяє популяризації використання криптовалюту у повсякденному житті.

Розробка програми для обміну криптовалют актуальна у зв'язку з зростаючою популярністю криптовалют та необхідністю створення безпечних та зручних інструментів для їх обміну.

Список ключових слів: ВЕБ-ОРІЄНТОВА ДОДАТОК, КРИПТОВАЛЮТА, ОБМІН КРИПТОВАЛЮТЬ, БАЗА ДАНИХ, ВЕБ-ДИЗАЙН, ІНТЕРФЕЙС КОРИСТУВАЧА, NODE.JS, REACT, DJANGO, REST API.

ABSTRACT

Explanatory note: 90 pages, 20 figures, 1 tables, 3 appendices, 20 sources.

Object of development: A web-oriented application for cryptocurrency exchange.

Purpose of the qualification work: Development of a web-oriented application for cryptocurrency exchange, ensuring convenience and security of exchange operations, using modern web development technologies and tools.

The introduction analyzes the current state of the problem, specifies the purpose of the qualification work and its application scope, justifies the relevance of the topic.

The first chapter analyzes the subject area, determines the relevance of the task and the purpose of the development, formulates the task definition, and specifies the requirements for the software implementation, technologies, and tools.

The second chapter analyzes existing solutions, selects the platform for development, and performs the design and development of the web-oriented application. It describes the operation of the application, the algorithm and structure of its functioning, as well as the invocation and loading of the application. It specifies the input and output data and characterizes the set of technical parameters.

The economic section determines the labor intensity of the developed information system, calculates the cost of developing the application, and estimates the time required for its creation.

The practical task involves creating a web-oriented application that provides a convenient and secure platform for cryptocurrency exchange, simplifies the exchange process, and promotes the use of cryptocurrency in daily life.

The development of a cryptocurrency exchange application is relevant due to the growing popularity of cryptocurrencies and the need to create secure and convenient tools for their exchange.

Keywords: WEB-ORIENTED APPLICATION, CRYPTOCURRENCY, CRYPTOCURRENCY EXCHANGE, DATABASE, WEB DESIGN, USER INTERFACE, NODE.JS, REACT, DJANGO, REST API.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

СУБД – система управління базами даних

MVC – Model View Controller

API - application programming interface

REST - representational state transfer

DRF – Django rest framework

SQL - structured query language

SHA – secure hash algorithm

JSON – JavaScript object notation

HTTP – hypertext transfer protocol

XML - eXtensible markup language

CSS - cascading style sheets

QR – quick response code

ВСТУП

В даний час інформаційні технології відіграють важливу роль у різних галузях та галузях, не виключаючи фінансові технології. Криптовалюти та пов'язані з ними технології охоплюють широкий спектр процесів, включаючи обмін валют, безпечні транзакції, управління активами, автоматизацію фінансових операцій, покращення користувальницького досвіду та оптимізацію обмінних курсів. У зв'язку зі швидким розвитком та зростаючою популярністю криптовалют, існує постійна потреба у нових розробках та технологіях, що сприяють підвищенню їхньої доступності.

У цій кваліфікаційній роботі розглядається розробка інформаційної системи криптообмінника, що включає як серверну так і клієнтську частини. Основна мета полягає у створенні зручного, безпечного та ефективного веб-додатку для обміну криптовалютами. Ідея полягає в тому, щоб надати користувачам можливість легко та швидко обмінювати криптовалюти, забезпечуючи високий рівень безпеки та зручність використання. Це може стати значним кроком у бік популяризації криптовалют та зростання їх використання у повсякденному житті.

Об'єктом дослідження є веб-орієнтований додаток для обміну криптовалютами, що забезпечує зручність обміну. Мета кваліфікаційної роботи полягає у розробці веб-орієнтованого додатку для обміну криптовалюти на готівку та навпаки.

Для досягнення поставленої мети необхідно вирішити ряд завдань, включаючи аналіз існуючих рішень у контексті криптовалютних обмінників, виявлення їх недоліків та переваг. Слід сформувати основні вимоги до розробки програми, що охоплюють функціональні характеристики, інформаційну безпеку, технічні засоби та програмну сумісність. Також важливо визначити функціональне призначення системи, що розробляється, і вибрати стек

технологій і мов програмування для створення програми. Потрібно спроектувати структуру, алгоритми та організацію вхідних та вихідних даних системи для загального розуміння її роботи та етапів розробки.

У процесі розробки необхідно створити серверну частину програми, що забезпечує надійне зберігання та обробку даних, а також клієнтську частину, яка буде надавати користувачам інтерфейс для взаємодії з системою. Також потрібно реалізувати інтеграцію із зовнішніми сервісами для отримання актуальних курсів криптовалют та забезпечення безпеки транзакцій.

Результатом даної кваліфікаційної роботи стане готовий до використання веб-додаток для обміну криптовалют, що задовольнятиме сучасним вимогам, зручності та ефективності. Ця програма зможе надати користувачам можливість безпечного та зручного обміну криптовалют, сприяючи їх ширшому використанню у повсякденному житті.

Також, фінальним етапом є розрахунок трудомісткості, вартості та витрат на розробку інформаційної системи. Це включає оцінку необхідних ресурсів, трудомісткість та час розробки.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКИ ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

Розробка інформаційних систем є складним процесом, що включає проектування, розробку, тестування та впровадження програмного забезпечення для збирання, зберігання, обробки та розповсюдження інформації. У контексті криптовалютних обмінників цей процес має свої особливості, пов'язані з безпекою, децентралізацією та специфікою роботи з цифровими активами.

Розробка інформаційних систем для криптовалютних обмінників є складним процесом, що включає кілька важливих етапів. Перший крок – це аналіз вимог, де збираються та аналізуються потреби користувачів, визначаються функціональні та нефункціональні вимоги до системи, а також проводиться аналіз ринку та конкурентів. На цьому етапі розуміють, які завдання має виконувати система, хто її цільова аудиторія і які функції потрібні.

Далі йде проектування системи, де розробляється архітектура та детально плануються її компоненти. Для криптообмінника це означає створення безпечної, масштабованої та надійної системи. Обираються технології та інструменти, які будуть використовуватись, такі як Django для бекенду та React для фронтенду.

Наступний етап – розробка системи, де розробники пишуть код відповідно до вимог та проектної документації. Вони створюють базу даних для зберігання інформації про користувачів та транзакції, реалізують API для інтеграції з платіжними шлюзами та блокчейн-мережами, додають функції обміну криптовалют, відстеження курсів, управління гаманцями тощо.

Після цього система проходить тестування для виявлення та виправлення помилок. Це включає різні типи тестів, такі як функціональні, навантажувальні,

інтеграційні та безпекові. Це особливо важливо для криптообмінника, де безпека та надійність є критичними аспектами.

Після успішного тестування система впроваджується у виробничу среду. Це означає розгортання програмного забезпечення на серверах, налаштування системи та навчання користувачів. Важливо забезпечити безперервність роботи та мінімізувати простої під час впровадження.

Після впровадження система потребує постійної підтримки та обслуговування. Це включає моніторинг роботи, виправлення помилок, оновлення програмного забезпечення та додавання нових функцій відповідно до змін ринку та вимог користувачів.

Розробка інформаційної системи для криптообмінника вимагає врахування специфіки роботи з цифровими активами, забезпечення високого рівня безпеки та відповідності законодавчим вимогам. Правильний підхід до проектування та реалізації такої системи дозволить створити надійний та зручний сервіс для обміну криптовалют.

Криптовалюти — це цифрові чи віртуальні валюти, що використовують криптографію для забезпечення безпеки транзакцій. Вони децентралізовані і зазвичай функціонують на основі блокчейну — розподіленої бази даних, яка зберігає запис транзакцій у вигляді блоків, пов'язаних між собою [1]. Першою та найбільш відомою криптовалютою є біткоїн, створений у 2009 році Сатоші Накамото [2]. Крім біткоїна, існує багато інших криптовалют, які називають альткоїнами, наприклад, Ethereum, Litecoin, Ripple.

Ключові принципи роботи криптовалют включають децентралізацію, прозорість та анонімність. Децентралізація означає, що відсутній центральний орган, який контролює мережу; натомість транзакції перевіряються учасниками мережі. Прозорість забезпечується тим, що всі транзакції публічно записуються в блокчейні, що робить їх неможливими для підробки. Анонімність дозволяє користувачам проводити транзакції без розкриття особистої інформації [1].

Процес обміну криптовалют може відбуватися через централізовані обмінники, такі як Binance чи Coinbase, де платформи виступають посередниками. Також існують децентралізовані обмінники (DEX), які дозволяють користувачам обмінювати криптовалюти безпосередньо між собою без посередників. Крім того, є крипто-готівкові обмінники, що спеціалізуються на обміні криптовалют на готівку і навпаки, діючи як онлайн, так і офлайн.

Основними викликами та ризиками у сфері криптовалют є висока волатильність (статистичний захід ступеня мінливості ціни активу протягом певного періоду часу), що призводить до значних коливань їхньої вартості. Це може бути ризикованим для інвесторів та користувачів, які здійснюють операції з криптовалютами.

Крім волатильності, існують і інші виклики, з якими стикається сфера криптовалют. Одним із них є питання безпеки. Криптовалютні обмінники та гаманці можуть стати мішенню для хакерів. Було багато випадків зламів, внаслідок яких користувачі втрачали свої кошти. Тому дуже важливо використовувати надійні платформи та дотримуватися правил безпеки, таких як двофакторна автентифікація та використання апаратних гаманців.

Регуляторні питання також є значним викликом. В різних країнах існують різні підходи до регулювання криптовалют, від повного заборони до вільного обороту. Це створює правову невизначеність для користувачів та компаній, що працюють із криптовалютами. Водночас регулятори прагнуть знайти баланс між захистом інвесторів та сприянням інноваціям у цій сфері.

Технологічний розвиток постійно удосконалює криптовалютні системи, але також приносити нові виклики. Наприклад, масштабованість мережі — здатність обробляти велику кількість транзакцій швидко та ефективно — є актуальною проблемою для багатьох криптовалют, включаючи біткоїн та Ethereum. Інженери працюють над вирішенням цих проблем шляхом

впровадження нових технологій, таких як Lightning Network для біткоїна або перехід на Ethereum 2.0 з використанням алгоритму Proof of Stake [1].

Обмін криптовалют на готівку та навпаки має свої особливості. Для цього часто використовуються спеціалізовані сервіси, що дозволяють здійснювати операції швидко та з мінімальними комісіями. Проте такі операції вимагають додаткових заходів безпеки, зокрема для запобігання шахрайству та відмиванню грошей.

У сучасному світі існує безліч систем для обміну криптовалют, кожна з яких пропонує свій набір функцій та переваг. Наприклад, Binance - одна з найбільших платформ для обміну криптовалют, яка відома своєю високою ліквідністю та великим вибором криптовалют. Вона також пропонує низькі комісії за транзакції та багатофункціональний інтерфейс, але може бути складною для новачків і іноді стикається із затримками при високому навантаженні.

Coinbase є популярною платформою, особливо серед новачків, завдяки своєму простому та інтуїтивно зрозумілому інтерфейсу (рис. 1.1.). Вона забезпечує високий рівень безпеки та пропонує безліч платіжних методів, але стягує вищі комісії та має обмежений вибір криптовалют у порівнянні з іншими платформами.

Kraken - одна з найстаріших криптовалютних бірж, відома своєю надійністю та високим рівнем безпеки. Вона пропонує низькі комісії та підтримку багатьох валютних пар, але її інтерфейс може здатися складним для новачків, а процес верифікації іноді займає багато часу.

Bitfinex пропонує широкий спектр інструментів та функцій для професійних трейдерів, а також високу ліквідність та низькі комісії для великих обсягів торгівлі. Однак її інтерфейс також складний для новачків, і платформа стикалася з проблемами регуляції та зломами у минулому.

KuCoin - це біржа, що швидко розвивається, яка пропонує широкий вибір криптовалют і низькі комісії. Вона регулярно впроваджує нові функції, але іноді стикається з проблемами ліквідності та повільною підтримкою клієнтів.

Кожна з цих платформ має свої сильні та слабкі сторони. Binance та Coinbase лідирують за обсягом торгівлі та кількістю користувачів, але мають свої складнощі з інтерфейсом та високими комісіями. Kraken та Bitfinex пропонують просунуті функції для досвідчених трейдерів, але можуть бути складними для новачків. KuCoin приваблює користувачів широким вибором криптовалют та низькими комісіями, але має проблеми з ліквідністю та підтримкою.

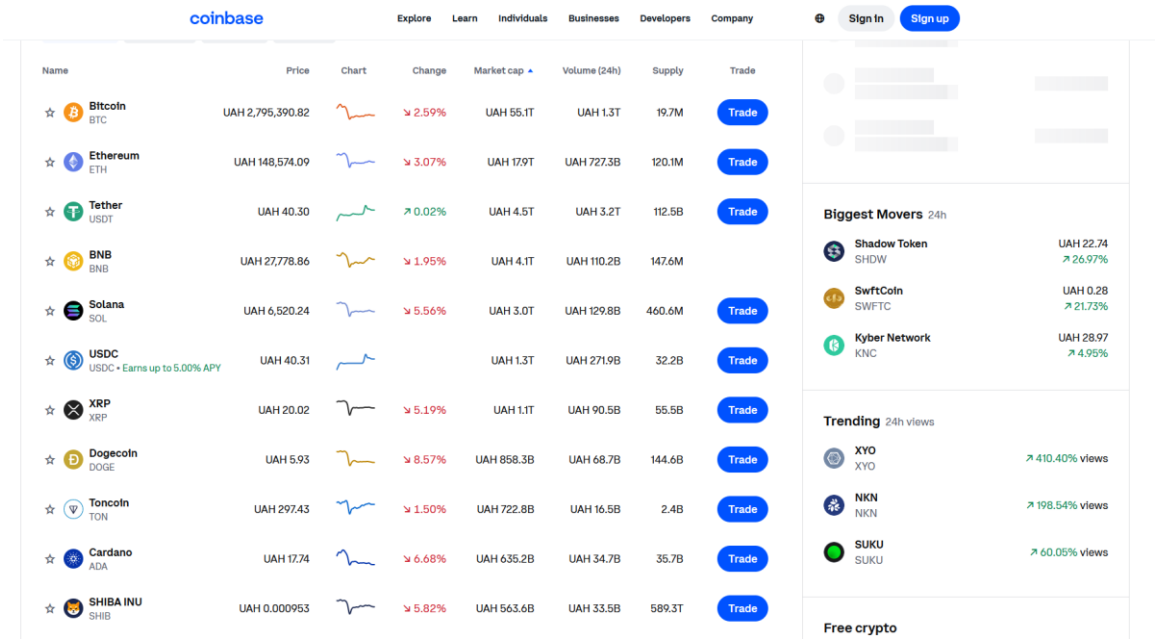

Рис. 1.1. Біржа Coinbase

1.2. Призначення розробки та галузь застосування

Метою розробки криптообмінника є створення веб-додатку, який дозволяє користувачам обмінювати готівку на криптовалюту. Криптовалюти, такі як біткойн, стають все більш поширеним засобом обміну, і водночас зростає попит на послуги, що полегшують їх обмін на традиційні фіатні валюти та готівку. Призначення такої розробки полягає у забезпеченні зручного та безпечного

механізму обміну між криптовалютами та готівкою, що може мати низку важливих областей застосування..

Область застосування такої програми може бути досить широкою. Воно можна використовувати як особистих, так комерційних цілей. Наприклад, люди, які хочуть інвестувати в криптовалюту, можуть використовувати цю програму для покупки криптовалюти. Крім того, компанії, які приймають до оплати криптовалюту, можуть використовувати цю програму для обміну криптовалюти на готівку.

Перша сфера застосування – це спрощення процесу обміну криптовалюти на готівку. Незважаючи на те, що криптовалюти все ще не є широко прийнятим засобом платежу, багато людей вважають за краще мати можливість обміняти свої криптовалютні активи на фіатні гроші. Розробка готівкових криптообмінників дає можливість користувачам обмінювати свої криптовалюти на готівку без необхідності використання біржі або інших складних процедур.

Друга сфера застосування – це забезпечення доступу до криптовалют для тих, хто воліє використовувати готівку. Незважаючи на те, що криптовалюти стають все більш популярними, багато людей все ще вважають за краще використовувати готівку через їх зручність або недовіру до цифрових форм оплати. Розробка готівкових криптообмінників дозволяє цим людям легко і безпечно купувати криптовалюти, надаючи їм альтернативний спосіб входу в світ цифрових активів.

Третя сфера застосування – це покращення фінансової інклюзії. Розробка готівкових криптообмінників може допомогти цим людям отримати доступ до цифрових фінансових послуг, таких як криптовалюти, без необхідності звернення до банків або інших фінансових інститутів.

Актуальністю для розробки криптообмінника готівки на криптовалюті є ряд факторів, що впливають на потребу в такому додатку та визначають його цільову аудиторію.

По-перше, це зростаюча популярність криптовалют та збільшення попиту на їх обмін на готівку. Все більше людей цікавиться криптовалютами і хоче інвестувати в них, а також використовувати їх для покупок та оплати послуг. У зв'язку з цим виникає потреба в надійних і зручних способах обміну криптовалют.

По-друге, існуючі обмінники криптовалют на готівку часто мають ряд недоліків, таких як низька безпека, відсутність підтримки різних валют та платіжних систем, складний та неінтуїтивний інтерфейс, а також високі комісії за обмін. У зв'язку з цим виникає потреба в розробці нових, більш надійних і функціональних додатків для обміну криптовалют.

По-третє, розробка криптообмінника може бути викликана бажанням створити новий бізнес у сфері криптовалют та надати користувачам зручний та безпечний спосіб обміну валют. Криптовалюти є одним із найбільш швидко зростаючих ринків у світі, і підприємці прагнуть знайти нові способи отримання вигоди з цього ринку.

Нарешті, по-четверте, розвиток технологій та поява нових інструментів та фреймворків для веб-розробки робить розробку криптообмінника більш доступною та ефективною. Сьогодні існує безліч готових рішень та бібліотек, які дозволяють швидко та якісно створювати веб-програми, включаючи обмінники криптовалют.

Таким чином, основою розробки криптообмінника готівки на криптовалюту і назад є ряд факторів, які впливають на потребу в такому додатку та визначають його цільову аудиторію.

1.3. Підстава для розробки

Відповідно до ОПП, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу. Тема роботи

узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

- ОПП за спеціальністю 122 «Комп'ютерні науки»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» №375-С від 23.04.2024 р.;
- завдання на кваліфікаційний проект «Розробка веб-застосунку для організації обміну криптовалют з використанням Django».

1.4. Постановка завдання

Розробка криптообмінника з динамічним оновленням курсів та зберіганням інформації про заявки на обмін - це складне та багатогранне завдання, що потребує комплексного підходу та ретельного проектування. Такий проект не тільки відображає актуальність та значущість криптовалют у сучасному світі, але й висуває високі вимоги до безпеки, продуктивності та зручності використання.

Розробка цієї системи складається з двох частин та включає наступні етапи:

1. Серверна частина:
    - аналіз та вибір технологій для розробки;
    - розробка бази даних;
    - створення моделей та серілізаторів;
    - розробка допоміжних функцій (динамічне оновлення курсів);
    - розробка інтерфейсів взаємодії з клієнтом (API);
    - тестування.

2. Клієнтська частина:
    - проектування та створення дизайну веб-додатка;
    - розробка функціональних компонентів;
    - розробка взаємодії з сервером;
    - тестування.

Крім того, необхідно враховувати можливість розширення функціоналу криптообмінника у майбутньому. Розробка модульної архітектури дозволить легко додавати нові можливості та інтегрувати додаткові сервіси, такі як підтримка нових криптовалютів, нових методів оплати або розширені можливості аналітики.

1.5. Вимоги до програмного виробу

1.5.1. Загальні вимоги

Вимоги до програмного продукту повинні забезпечувати простоту та зручність використання, а також масштабованість.

Простота та зручність використання є ключовими факторами, які впливають на популярність та успіх веб-додатків. Користувачі хочуть, щоб програма була легко зрозумілою і не вимагала великої кількості часу та зусиль для освоєння. Тому програма повинна мати простий і інтуїтивний інтерфейс, який буде зрозумілий навіть тим користувачам, які не мають великого досвіду роботи з криптовалютами або веб-додатками.

Масштабованість також є важливою вимогою до програмного продукту. Додаток має бути здатним обробляти велику кількість запитів і транзакцій, а також підтримувати велику кількість користувачів. Це означає, що програма має бути спроектована таким чином, щоб її можна було легко масштабувати відповідно до зростаючих потреб.

Програмний продукт повинен забезпечувати надійність та безпеку. Додаток має бути надійним і працювати без збоїв та помилок, а також забезпечувати безпеку транзакцій та даних користувачів. Для цього необхідно використовувати сучасні технології та інструменти, а також проводити регулярні тестування та аудити безпеки.

1.5.2. Вимоги до функціональних характеристик

Вимоги до функціональних характеристик програмного продукту для API, а також для веб-додатків з інтеграцією цього API:

- Підтримка різних криптовалют: програма має підтримувати широкий вибір криптовалют, включаючи Bitcoin, Ethereum, Litecoin та інші.
- Динамічне оновлення курсів: програма повинна забезпечувати динамічне оновлення курсів криптовалют у реальному часі, щоб користувачі могли отримати актуальну інформацію про курси.
- Зберігання інформації про заявки на обмін: програма повинна зберігати інформацію про заявки на обмін, включаючи суму, валюту, курс та статус заявки.
- Створення та відстеження заявок на обмін: користувачі повинні мати можливість створювати заявки на обмін криптовалют та відстежувати їх статус у додатку.
- Масштабованість: додаток повинен бути спроектований таким чином, щоб його можна було легко масштабувати відповідно до зростаючих потреб.
- Безпека та надійність: програма повинна забезпечувати безпеку та надійність транзакцій та даних користувачів, використовуючи сучасні технології та інструменти.

- Простий та інтуїтивний інтерфейс: додаток повинен мати простий та інтуїтивний інтерфейс, який зрозумілий навіть тим користувачам, які не мають великого досвіду роботи з криптовалютами або веб-додатками.

В цілому, вимоги до функціональних характеристик програмного продукту для API криптообмінника та веб-додатка повинні забезпечувати широкий вибір криптовалют, динамічне оновлення курсів, зберігання інформації про заявки на обмін, створення та відстеження заявок на обмін, інтеграцію з платіжними системами, масштабованість, безпеку та надійність , а також простий та інтуїтивний інтерфейс. Це дозволить створити додаток, який матиме попит і відповідатиме всім вимогам користувачів та бізнесу.

1.5.3. Вимоги до інформаційної та програмної сумісності

Вимоги до інформаційної та програмної сумісності програмного продукту включають:

- Сумісність з різними операційними системами: програма повинна бути сумісною з різними операційними системами, включаючи Windows, macOS та Linux.
- Сумісність із різними браузерами: веб-додаток має бути сумісним із різними браузерами, включаючи Google Chrome, Mozilla Firefox, Safari та Microsoft Edge.
- Сумісність з різними пристроями: веб-програма має бути адаптивною та сумісною з різними пристроями, включаючи настільні комп'ютери, ноутбуки, планшети та смартфони.
- Інтеграція з іншими додатками та сервісами: програма має бути спроектована таким чином, щоб її можна було легко інтегрувати з іншими додатками та сервісами, включаючи платіжні системи, CRM-системи та системи обліку.

- Використання стандартних протоколів та форматів даних: програма повинна використовувати стандартні протоколи та формати даних, такі як HTTPS, JSON та XML, для забезпечення сумісності з іншими програмами та сервісами.

Загалом, вимоги до сумісності веб-додатку повинні забезпечувати сумісність із різними операційними системами, браузерами та пристроями, а також легкість інтеграції з іншими програмами та сервісами. Це дозволить створити додаток, який буде зручно та ефективно використовувати для широкого кола користувачів.

1.5.4. Вимоги до складу та параметрів технічних засобів

Вимоги до складу та параметрів технічних засобів для розробки криптообмінника з динамічним оновленням курсів та зберіганням інформації про заявки на обмін залежать від масштабу проекту, очікуваного навантаження та вибраних технологій. Необхідно вибрати хостинг-провайдер або хмарний сервіс, здатний забезпечити високу продуктивність та надійність, з можливістю масштабування ресурсів у міру зростання навантаження.

Для зберігання інформації про користувачів, заявки на обмін та інші дані необхідно вибрати відповідну базу даних, рекомендується використовувати реляційні бази даних, з можливістю масштабування та реплікації даних для забезпечення високої доступності та відмовостійкості.

Для зберігання криптовалютних активів та забезпечення їх безпеки необхідно використовувати криптографічні гаманці та спеціалізовані сервіси зберігання, такі як холодні гаманці (cold wallets) або мультипідписні гаманці (multisig wallets).

Для забезпечення високої продуктивності та надійності необхідно використовувати сучасні мережі та протоколи, такі як HTTP/2 для прискорення передачі даних та SSL/TLS для забезпечення безпеки з'єднань.

Для забезпечення безпеки програми необхідно використовувати засоби захисту від DDoS-атак та зломів, такі як фільтри трафіку, веб-брандмауери та системи виявлення вторгнень (IDS/IPS).

Для забезпечення безпеки даних та можливості швидкого відновлення у разі збоїв необхідно використовувати системи резервного копіювання та відновлення даних.

Залежно від обсягу даних та очікуваного навантаження необхідно вибрати серверне обладнання або хмарні ресурси з достатнім обсягом процесорної потужності та оперативної пам'яті для забезпечення швидкої обробки запитів та високої продуктивності програми.

Важливо забезпечити захист даних користувачів та конфіденційність інформації. Для цього необхідно використовувати сучасні методи шифрування даних та захисту від витоку інформації.

Для забезпечення безпеки даних та можливості швидкого відновлення у разі збоїв необхідно регулярно створювати резервні копії даних та використовувати системи автоматичного відновлення. Відповідність вимогам безпеки та законодавству: Необхідно враховувати вимоги щодо безпеки даних та відповідати законодавству в галузі обробки персональних даних та фінансових операцій, таких як GDPR або PCI DSS.

Вимоги до складу та параметрів технічних засобів для розробки криптообмінника повинні бути узгоджені з вимогами до продуктивності, надійності, безпеки та масштабованості програми.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Веб-додаток служить для забезпечення зручного, швидкого та безпечного процесу обміну криптовалютами на готівку та назад. Воно включає стартове вікно з вибором валют для обміну, де користувачеві відображаються списки, що випадають, для вибору вихідної і цільової валют, при цьому курси валют оновлюються в реальному часі. Система регулярно отримує та оновлює інформацію про курси криптовалют із зовнішніх джерел, таких як біржі криптовалют, що дозволяє користувачам отримувати найактуальнішу інформацію.

Після вибору валют користувачі вводять необхідні дані до створення заявки на обмін, інтерфейс може бути інтуїтивно зрозумілим і забезпечувати коректність введення даних. Після введення всіх даних користувач підтверджує заявку на обмін, програма розраховує підсумкову суму з урахуванням комісій та відображає користувачеві для остаточного підтвердження.

При підтвердженні заявки створюється унікальний та секретний код, який користувач повинен надати касиру для завершення операції. Система генерує унікальний ідентифікатор заявки та секретний код, які відображаються користувачеві на екрані та можуть бути надіслані на вказану контактну адресу. Користувач може відстежувати статус заявки через веб-додаток, а також отримувати повідомлення про зміну статусу заявки через вибрані канали. Для касира передбачено спеціальний інтерфейс, що дозволяє перевіряти коди заявок та секретні коди, а також підтверджувати отримання та передачу коштів.

2.2. Опис застосованих математичних методів

У проекті розробки криптообмінника основна увага приділялася інтеграції готових рішень та технологій для забезпечення функціональності та безпеки системи. Не використовувалися складні математичні методи чи моделі, такі як інтерполяція, регресійний аналіз чи методи оптимізації. У проекті використовувалися перевірені та надійні готові криптографічні алгоритми та стандарти.

Одним із таких алгоритмів, використаних у проекті, був SHA-1. SHA-1 є криптографічної хеш-функцією, що перетворює вхідне повідомлення довільної довжини фіксоване хеш-значення довжиною 160 біт. Хоча SHA-1 в даний час вважається застарілим і менш безпечним порівняно з сучаснішими алгоритмами, такими як SHA-256, він більше підходить у цьому проекті, де його безпека вважається достатньою.

Процес хешування SHA-1 включає розбиття повідомлення на блоки по 512 біт, доповнення повідомлення до потрібної довжини, ініціалізацію з використанням фіксованих початкових значень, а потім ітеративну обробку кожного блоку із застосуванням операцій побітового зсуву і логічних функцій.

Таким чином, у проекті криптообмінника основна увага була приділена інтеграції готових криптографічних алгоритмів та стандартів, таких як SHA-1 для забезпечення базової функціональності та безпеки системи. Більш складні математичні методи та моделі не використовувалися, оскільки основний наголос був зроблений на використання перевірених рішень та стандартних практик.

2.3. Опис використаної архітектури та шаблонів проектування

Архітектура системи криптообмінника включає кілька ключових компонентів, що забезпечують функціональність, надійність і безпеку. Основна

увага приділяється поділу логіки програми, керуванню даними та уявленню інтерфейсу. У серверній частині системи застосовується шаблон проектування MVC (Model-View-Controller).

Клієнтська частина (Frontend) включає в себе інтерфейс користувача (UI), що відповідає за відображення інформації користувачеві і взаємодію з ним. Включає елементи для вибору валют, введення даних та відстеження стану заявок. Взаємодія з сервером здійснюється асинхронно для оновлення даних без перезавантаження сторінки, що дозволяє динамічно отримувати актуальні курси валют та інформацію про статус заявок.

Серверна частина (Backend) включає шаблон проектування MVC. Model (Модель) визначає структуру даних та відносини між ними, включаючи моделі для подання користувачів, заявок на обмін та курс валют. Також опрацьовує бізнес-логіку, пов'язану з обміном валют, розрахунками комісій та управлінням заявками. View (Подання) відповідає за формування та подання даних користувачеві, отримує дані з моделі та відображає їх у вигляді HTML-сторінок або JSON-відповідей для клієнтської частини. Шаблони, що використовуються, представлені для динамічного створення інтерфейсу користувача на основі даних, отриманих з моделей. Controller (Контролер) приймає запити від клієнта, обробляє їх і передає відповідні моделі для виконання бізнес-логіки. Також управляє взаємодією між користувачем, моделями і уявленнями, забезпечуючи коректну обробку даних, і повернення результатів користувачеві.

База даних використовується для надійного зберігання та управління даними користувачів, заявок на обмін та історичних курсів валют. Оптимізовані запити та індекси забезпечують швидке вилучення та оновлення даних, що важливо для динамічного оновлення курсів та обробки заявок.

API забезпечує взаємодію між клієнтською та серверною частинами через стандартизовані HTTP-запити. API використовується для отримання курсів валют, створення заявок на обмін та відстеження їхнього стану. Включає

механізми автентифікації та авторизації, щоб гарантувати безпеку передачі даних та доступ лише для авторизованих користувачів. Використовуються методи захисту від поширених атак, таких як SQL-ін'єкції, XSS та CSRF.

На закінчення, використання системи криптообмінника з шаблоном проектування MVC дозволяє чітко розділити відповідальність між різними компонентами програми, забезпечуючи модульність, розширюваність і легкість підтримки. Клієнтська частина забезпечує зручну взаємодію з користувачем, серверна частина обробляє запити та виконує бізнес-логіку, а база даних надійно зберігає всі необхідні дані.

2.4. Опис використаних технологій та мов програмування

2.4.1. Мова Python

Python – одна з найбільш популярних та затребуваних мов програмування у світі. Створений Гвідо ван Россумом наприкінці 1980-х років, Python швидко завоював серця розробників своїм простим і зрозумілим синтаксисом, який робить його доступним для початківців, але при цьому потужним та гнучким для досвідчених програмістів [5].

Однією з ключових особливостей Python є його читання. Синтаксис Python розроблений таким чином, щоб код на ньому виглядав майже як звичайний англійський текст, що робить його легко читати навіть для тих, хто тільки починає вивчати програмування. Це дозволяє розробникам швидко розуміти та модифікувати код, а також співпрацювати над проектами.

Ще однією перевагою Python є його широка стандартна бібліотека. Python поставляється з великою кількістю модулів та інструментів, які дозволяють виконувати широкий спектр завдань, починаючи від обробки тексту та роботи з базами даних, і до створення веб-додатків та машинного навчання. Завдяки цій

великій бібліотеці розробники можуть швидко розпочати роботу над своїми проектами, не витрачаючи час на написання коду з нуля [5].

Python також відомий своєю універсальністю та популярністю в різних галузях, таких як веб-розробка, дослідження, аналіз даних, розробка ігор, автоматизація завдань та багато іншого. Ця мова програмування використовується такими великими компаніями, як Google, Facebook, Instagram, Dropbox та багатьма іншими, що підкреслює його значущість та вплив у сучасному світі технологій.

Python є основною мовою програмування для багатьох популярних веб-фреймворків, таких як Django, Flask, та Pyramid. Ці фреймворки надають розробникам готові інструменти та структури для створення веб-застосунків більш ефективно.

Python також широко застосовується в обробці HTTP-запитів та взаємодії з веб-серверами через бібліотеки, такі як Requests. Це дозволяє розробникам створювати клієнт-серверні веб-додатки та працювати із зовнішніми API.

Велика кількість бібліотек та інструментів Python полегшує роботу з базами даних, обробкою форм, автоматизацією тестування та багатьма іншими завданнями, пов'язаними з веб-розробкою.

Python також використовується для створення сценаріїв та скриптів, які автоматизують різні аспекти розгортання та управління веб-додатками, такі як розгортання на серверах, резервне копіювання даних та моніторинг продуктивності.

Python відіграє ключову роль у розробці завдяки своїй простоті, потужним інструментам та широкій спільноті розробників.

2.4.2. RESTful API

REST API забезпечуючи взаємодію та обмін даними між різними програмними системами. В основі концепції REST API лежать принципи та практики, які полегшують обмін даними між клієнтом та сервером [6].

Центральне значення для концепції REST API має його архітектурний стиль, який наголошує на використанні стандартних методів HTTP, таких як GET, POST, PUT та DELETE для виконання операцій з ресурсами. Цей уніфікований інтерфейс спрощує комунікацію між клієнтом та сервером, сприяючи чіткому поділу обов'язків та підвищуючи загальну ефективність веб-додатків [6].

Однією з ключових переваг REST API є його стан «без стану», коли кожен запит від клієнта до сервера містить всю необхідну інформацію для обробки запиту. Ця відсутність стану не тільки спрощує реалізацію сервера, але й покращує надійність та масштабованість, усуваючи необхідність керування сеансами на стороні сервера.

Більше того, REST сприяє ресурсно-орієнтованому підходу до проектування веб-служб, при якому дані представлені як ресурси з унікальними ідентифікаторами (URI). Дотримуючись принципів REST, розробники можуть створювати API, які інтуїтивно зрозумілі, гнучкі і легко розширюються, що дозволяє адаптуватися до бізнес-вимог, що змінюються, і потреб користувачів.

REST забезпечує слабку зв'язок між компонентами клієнта та сервера, дозволяючи їм еволюціонувати незалежно без порушення загальної функціональності системи. Цей поділ сприяє модульності та повторному використанню коду, дозволяючи організаціям створювати складні системи, що складаються з менших взаємопов'язаних компонентів.

Крім того, REST API відіграє ключову роль у забезпеченні інтеграції та взаємодії різних програмних систем на різних платформах та пристроях. Надаючи стандартизований підхід до обміну даними, REST API сприяє

бездоганній інтеграції між різнорідними системами, стимулюючи співпрацю та інновації в цифровій екосистемі.

Важливо також підкреслити вплив REST API на розвиток сучасної екосистеми додатків. Завдяки стандартизованому підходу до веб-розробки різні сервіси можуть легко інтегруватися один з одним, утворюючи комплексні системи, здатні забезпечити ширший функціонал і задовольнити різноманітні потреби користувачів.

2.4.3. Django REST Framework

У розробці криптообмінника використовувався Django REST Framework (DRF), розширення фреймворку Django, призначене для створення веб-сервісів API. DRF надає потужні інструменти та принципи для гнучкої та швидкої розробки RESTful API на базі Django. Включення DRF у проект дає доступ до корисних функцій та абстракцій, які спрощують створення та підтримку веб-сервісів.

DRF надає серіалізатори, що перетворюють дані моделей Django у формат JSON та назад, полегшуючи передачу даних між сервером та клієнтом. Класи уявлень DRF автоматично обробляють HTTP-методи (GET, POST, PUT, DELETE тощо) і відповідні їм дії, спрощуючи створення точок входу API. DRF також надає готові засоби для реалізації аутентифікації та авторизації користувачів API, такі як базова автентифікація, токени, OAuth та багато іншого [9].

DRF дозволяє надсилати та приймати дані у різних форматах, таких як JSON, XML, YAML та інших, роблячи API більш гнучким та універсальним. Він надає інструменти для реалізації пагінації та фільтрації результатів запитів API, дозволяючи ефективно управляти обсягом даних та забезпечувати чуйність API. DRF автоматично генерує документацію API на основі коду, що полегшує розуміння та використання API розробниками [9].

Використання Django REST Framework у серверній частині криптообмінника дозволило створити потужне та гнучке веб-сервіс API з мінімальними зусиллями. Він забезпечує надійність, безпеку та ефективність при роботі з даними та взаємодії з клієнтськими додатками. Завдяки гнучкості та простоті використання, DRF залишається одним з найбільш популярних інструментів для створення API на основі Django.

2.4.4. JavaScript та React

JavaScript - одна з найпоширеніших та найвпливовіших мов програмування в сучасному світі. На початку свого шляху JavaScript було створено Бренданом Ейхом у компанії Netscape у 1995 році. Спочатку він замислювався як простий мову сценаріїв для пожвавлення веб-сторінок, щоб зробити їх більш інтерактивними. На той час веб був статичним і обмеженим у функціоналі, і JavaScript привніс довгоочікувану динаміку, дозволяючи створювати анімації, перевіряти форми, реагувати на дії користувача без необхідності перезавантаження сторінки [11].

З роками JavaScript еволюціонував з простого інструменту для фронтенд-розробки в повноцінну мову програмування, що застосовується в різних областях. Одним із ключових моментів у його розвитку стала поява середовища виконання Node.js, розробленого Райаном Далем у 2009 році. Node.js дозволив розробникам використовувати JavaScript на серверній стороні, що відкрило нові горизонти для створення повноцінних веб-застосунків, працюючи з однією технологією і на фронтенді, і на сервері.

Сьогодні JavaScript продовжує залишатись на передньому краї веб-розробки. Його вплив видно всюди: від простих сайтів до складних веб-додатків, від настільних до серверних рішень. Він став невід'ємною частиною повсякденного життя розробників та користувачів по всьому світу.

У проекті JavaScript з React використовувався для створення інтерфейсу користувача, взаємодії з сервером через API та забезпечення динамічного оновлення даних. Завдяки простоті використання, гнучкості та продуктивності, React став одним з найбільш популярних інструментів у веб-розробці, забезпечуючи створення сучасних і чуйних інтерфейсів.

React заснований на компонентному підході, що дозволяє розбити інтерфейс користувача на невеликі незалежні компоненти. Кожен компонент відповідає за свою логіку та подання даних, що полегшує розробку, тестування та повторне використання коду [12]. React використовує віртуальний DOM для оптимізації продуктивності під час оновлення інтерфейсу. Він дозволяє ефективно оновлювати лише ті частини сторінки, які змінилися, мінімізуючи кількість операцій рендерингу та підвищуючи чуйність програми.

React надає зручний спосіб керування станом програми за допомогою внутрішнього стану компонентів та передачі даних через пропси. Це робить код більш передбачуваним та легко підтримуваним. JavaScript та React підтримують модульність та розширюваність коду, що дозволяє розробникам використовувати сторонні бібліотеки та компоненти для швидкої розробки та додавання нових функцій.

Навколо React сформувалася багата екосистема інструментів та бібліотек, таких як Redux для керування станом, React Router для навігації, Axios для роботи з HTTP-запитами та багато інших.

У цьому проекті використовувалася бібліотека Axios. Axios - це бібліотека для виконання HTTP-запитів із браузера або Node.js, яка забезпечує простий та зручний спосіб взаємодії із сервером.

Використання Axios у проекті дозволило ефективно взаємодіяти із сервером, надсилати запити для отримання курсів валют, створення та управління заявками на обмін. Завдяки простому та зручному інтерфейсу, Axios

став популярним інструментом у веб-розробці для роботи з HTTP-запитами, забезпечуючи надійну та ефективну передачу даних між клієнтом та сервером.

2.5. Опис структури програми та алгоритмів її функціонування

2.5.1. Опис серверної частини

2.5.1.1. База даних

База даних для цього проекту складається з трьох таблиць (рис. 2.1). Перша таблиця містить інформацію про валюти, що використовуються у програмі. Для кожної валюти зберігається її назва, код, флаг, який вказує, чи є валюта фіатною, посилання на зображення та поточний курс у доларах. Для криптовалют також зберігається адреса гаманця.

Друга таблиця містить інформацію про замовлення, створені користувачами програми. Для кожного замовлення зберігається унікальний ідентифікатор, код, номер телефону та електронна пошта користувача, валюта, з якої здійснюється обмін, та валюта, в яку здійснюється обмін. Також зберігається сума обміну в обох валютах та адреса гаманця для криптовалюту.

Третя таблиця містить інформацію про мінімальну та максимальну суму обміну в доларах. Ця інформація використовується для перевірки суми обміну під час створення замовлення.

Взаємозв'язок між таблицями здійснюється з допомогою зовнішніх ключів. Друга таблиця має два зовнішні ключі, що зв'язують її з першою таблицею. Ці ключі вказують на валюти, з яких і які здійснюється обмін.

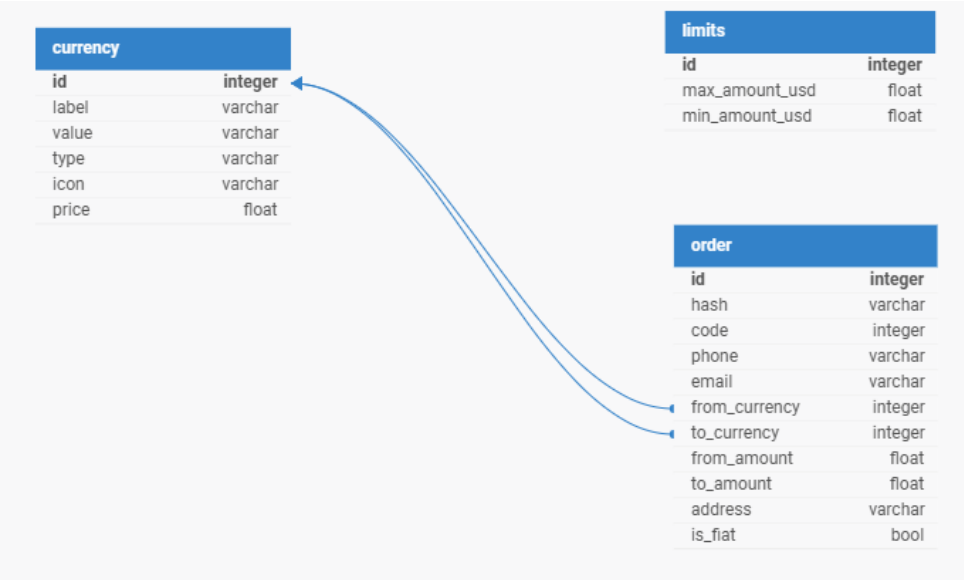

Рис. 2.1. Структура БД

2.5.1.2. Моделі та серіалізація

Для реалізації взаємодії з базою даних у проекті було створено моделі Django (рис. 2.2). Моделі Django є класами, які описують структуру таблиць бази даних і поля, які вони містять.

Моделі, які були створені для даного проекту, є класами, що описують структуру таблиць бази даних, які використовуються в додатку. Наприклад, модель «Currency» описує структуру таблиці, яка зберігає інформацію про валюти, які у додатку. Модель «Order» описує структуру таблиці, яка зберігає інформацію про замовлення, створені користувачами програми. Модель «Limit» описує структуру таблиці, яка зберігає інформацію про мінімальну та максимальну суму обміну в доларах.

Моделі Django також надають методи роботи з базою даних, такі як створення, читання, оновлення та видалення записів. Ці методи використовуються в програмному коді для взаємодії з базою даних.

Використання моделей Django дозволяє спростити роботу з базою даних, робить код більш читабельним та зручним.

```
10 usages
class Currency(models.Model):
    label = models.CharField(max_length=20)
    value = models.CharField(max_length=5)
    is_fiat = models.BooleanField()
    icon = models.CharField(max_length=100)
    price = models.FloatField()
    address = models.CharField(max_length=256, null=True)

    def __str__(self):
        return self.label


6 usages
class Order(models.Model):
    hash = models.CharField(max_length=256)
    code = models.IntegerField()
    phone = models.CharField(max_length=16)
    email = models.CharField(max_length=100)
    from_currency = models.ForeignKey(Currency, on_delete=models.CASCADE, related_name='from_currency')
    to_currency = models.ForeignKey(Currency, on_delete=models.CASCADE, related_name='to_currency')
    from_amount = models.FloatField()
    to_amount = models.FloatField()
    address = models.CharField(max_length=256, null=True)


4 usages
class Limit(models.Model):
    min_amount_usd = models.FloatField()
    max_amount_usd = models.FloatField()
```

Рис. 2.2. Код Django моделей

Моделі, які були створені для цього проекту, використовують SQLite як базу даних. Під час створення моделей Django автоматично генерується структура таблиць SQLite, яка відповідає структурі моделей.

Для реалізації взаємодії між клієнтською частиною програми, написаною на React, та серверною частиною, написаною на Django, були реалізовані серіалізація та десеріалізація моделей.

Серіалізація та десеріалізація моделей є важливими аспектами розробки веб-додатків, оскільки вони дозволяють обмінюватися даними між клієнтською та серверною частинами програми. У нашому проекті ми використовували Django REST framework для реалізації серіалізації та десеріалізації моделей.

Django REST framework пропонує набір інструментів для створення RESTful API, включаючи серіалізатори. Серіалізатори дозволяють перетворювати об'єкти моделі на формат JSON і назад. Були створені серіалізатори для моделей «Currency», «Order» та «Limit» (рис. 2.3), які дозволяють перетворювати об'єкти моделі у формат JSON.

Серіалізатори також дозволяють настроювати формат серіалізованих даних. Наприклад, ми можемо включати чи виключати певні поля моделі із серіалізованих даних. Це дозволяє гнучко настроювати формат даних, який передається між клієнтською та серверною частинами програми.

Серіалізація та десеріалізація моделей також забезпечують гнучкість та масштабованість програми. Ми можемо легко додавати нові функції та змінювати структуру даних, не переписуючи весь код програми. Крім того, серіалізація та десеріалізація моделей дозволяють використовувати різні формати даних, такі як XML або YAML, для обміну даними між клієнтською та серверною частинами програми.

```python
class OrderCreateSerializer(serializers.ModelSerializer):
    address = serializers.CharField(required=False, allow_blank=True, allow_null=True)

    class Meta:
        model = Order
        fields = ['phone', 'email', 'from_currency', 'to_currency', 'from_amount', 'address']

    1 usage (1 dynamic)
    def create(self, validated_data):
        order_code = utils.get_random_code()
        order_hash = utils.get_hash(validated_data['phone'], order_code)
        order_amount = (validated_data['from_amount'] *
                        validated_data['from_currency'].price / validated_data['to_currency'].price)

        decimal_places = 5
        if validated_data['to_currency'].is_fiat:
            decimal_places = 2

        order = Order.objects.create(
            hash=order_hash,
            code=order_code,
            to_amount=round(order_amount, decimal_places),
            **validated_data
        )

        return order
```

Рис. 2.3. Приклад серіалізатора моделі заявок

2.5.1.3. Динамічне оновлення курсів

Для забезпечення динамічного оновлення курсів криптовалют у додатку було створено рішення, яке ґрунтується на використанні API на біржі Binance Для реалізації цього функціоналу було розроблено спеціальну функцію, яка надсилає GET-запит на API Binance та отримує поточні ціни на криптовалюти у форматі JSON (рис. 2.4).

Для того, щоб відфільтрувати криптовалюти, які не є фіатними, було використано фільтрацію моделі «Currency». Були отримані символи всіх криптовалют, які не є фіатними, та сформовано список символів для запиту до API Binance.

Для запобігання занадто частому оновленню курсів було додано перевірку часу останнього оновлення. Якщо минуло менше однієї хвилини від останнього оновлення, функція оновлення не виконується.

Завдяки цьому рішенню наша програма завжди має актуальні курси криптовалют, отримані з біржі Binance. Це дозволяє користувачам програми завжди бути в курсі поточних цін на криптовалюти та здійснювати обміни на основі актуальної інформації.

Після отримання поточних цін на криптовалюти з біржі Binance, було здійснено їх запис до бази даних. Для цього було використано модель, яка була оновлена з новими цінами.

```
1 usage
def get_prices(symbols):
    for x in range(len(symbols)):
        symbols[x] = symbols[x] + 'USDT'

    api_request = ('https://api.binance.com/api/v3/ticker/price?symbols=' +
                   json.dumps([str(x) for x in symbols]).replace( __old: ' ', __new: ''))
    response = requests.get(api_request)
    if response.status_code == 200:
        return json.loads(response.text)
    return None


1 usage
def update():
    global last_update
    if last_update is not None:
        current_time = timezone.now()
        time_difference = current_time - last_update
        minutes = time_difference.total_seconds() // 60

        if minutes < 1:
            return

    coins = Currency.objects.filter(is_fiat=False)
    symbols = [record.value for record in coins]

    prices = get_prices(symbols)

    if prices is not None:
        for x in prices:
            symbol = x['symbol'].replace('USDT', '')
            coin = Currency.objects.get(value=symbol)

            coin.price = float(x['price'])
            coin.save()

        last_update = timezone.now()
```

Рис. 2.4. Код для оновлення курсів

2.5.1.4. Інтерфейси взаємодії

Інтерфейси взаємодії є набором функцій, які надають API для взаємодії з додатком. У цьому проекті були створені інтерфейси для отримання інформації про обмін валют, створення замовлення на обмін та отримання інформації про замовлення.

Один з інтерфейсів надає інформацію про доступні криптовалюти та фіатні валюти, а також обмеження на обмін (рис. 2.5). Він надсилає запит на

оновлення курсів валют та повертає список криптовалют, список фіатних валют та обмеження на обмін. При отриманні запиту від клієнта функція обробляє його та повертає потрібну інформацію у форматі JSON.

```python
@api_view(['GET'])
def get_exchange_options(request):
    prices.update()

    limits = Limit.objects.last()
    crypto_currencies = Currency.objects.filter(is_fiat=False)
    fiat_currencies = Currency.objects.filter(is_fiat=True)
    response_data = {
        "limits": LimitsSerializer(limits).data,
        "give": CurrencySerializer(crypto_currencies, many=True).data,
        "get": CurrencySerializer(fiat_currencies, many=True).data
    }

    return Response(response_data, status=status.HTTP_200_OK)
```

Рис. 2.5. Отримання інформації про доступні валюти

Інший інтерфейс створює нове замовлення обмін валют (рис. 2.6). Він приймає інформацію про замовлення, таку як номери телефонів, адреси електронної пошти, суми та типи валют і створює нові замовлення. При отриманні запиту від клієнта функція перевіряє введені дані та створює новий замовлення в базі даних. Після створення замовлення функція повертає хеш-код замовлення у форматі JSON.

```python
@api_view(['POST'])
def add_order(request):
    data = request.data
    serializer = OrderCreateSerializer(data=data)
    if serializer.is_valid():
        order = serializer.save()
        return Response( data: {
            'hash': order.hash
        }, status=status.HTTP_200_OK)
    return Response(serializer.errors, status=status.HTTP_200_OK)
```

Рис. 2.6. Додавання нового замовлення

Третій інтерфейс дає інформацію про замовлення (Рис. 2.7). Він приймає хеш-код замовлення та повертає інформацію про замовлення, включаючи статус,

суми та типи валют. При отриманні запиту від клієнта функція шукає замовлення бази даних і повертає потрібну інформацію у форматі JSON.

```
@api_view(['GET'])
def get_order(request, order_hash):
    order = Order.objects.filter(hash=order_hash)
    if order.exists():
        return Response(OrderReadSerializer(order.last()).data, status.HTTP_200_OK)

    return Response(status=status.HTTP_404_NOT_FOUND)
```

Рис. 2.7. Отримання інформації про замовлення

Ці методи надають API для взаємодії з додатком та дозволяють клієнту отримувати потрібну інформацію та створювати замовлення на обмін валют. Вони також забезпечують безпеку та надійність передачі даних, оскільки використовують протокол HTTPS для шифрування інформації.

2.5.1.5. Тестування запитів

Для тестування API використовувався інструмент Postman. Postman - це популярна програма для тестування API, яка дозволяє відправляти HTTP-запити на сервер та отримувати відповіді. Воно надає зручний графічний інтерфейс для створення та надсилання запитів, а також для перегляду отриманих відповідей.

У Postman можна створювати колекції запитів, які можна зберігати та повторно використовувати. Це дозволяє економити час та спрощує процес тестування. Крім того, Postman надає можливість додавати параметри та заголовки до запитів, а також тестувати різні типи автентифікації.

При тестуванні API за допомогою Postman можна перевіряти роботу різних методів HTTP, таких як GET, POST, PUT, DELETE та інші. Це дозволяє перевіряти роботу API у різних сценаріях та переконатися, що воно працює корректно.

Postman також дає можливість настроювати тести та перевіряти автоматично отримані відповіді. Це дозволяє автоматизувати процес тестування та зменшити кількість ручної роботи.

Наприклад, відправимо POST-запит на створення нового замовлення на обмін валют (рис. 2.8). У тілі запиту були передані необхідні дані, такі як тип валюти, яку хочуть обміняти, тип валюти, яку хочуть обміняти, сума обміну, адресу електронної пошти, номер телефону та адресу гаманця для отримання криптовалюти.

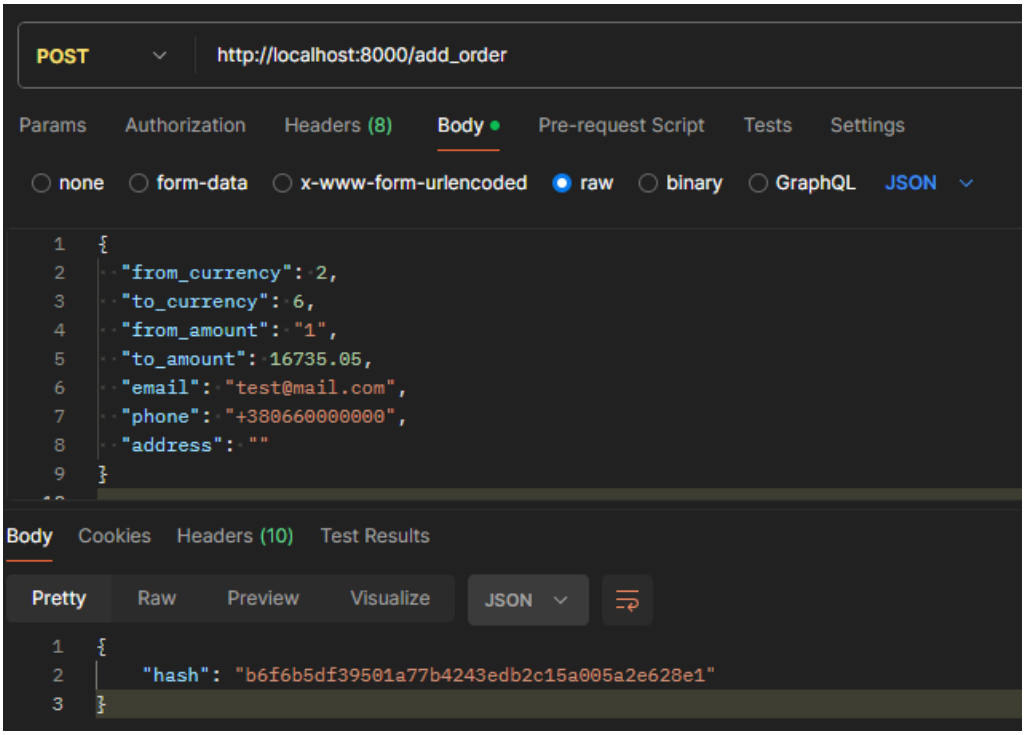

Рис. 2.8. Відправка POST запиту на додавання нового замовлення

У відповідь на запит було отримано об'єкт JSON, що містить хеш-код створеного замовлення. Цей хеш-код можна використовувати для відстеження статусу замовлення та отримання додаткової інформації про нього.

Тепер відправимо GET запит з хешем заявки для отримання інформації про неї (рис. 2.9). У відповідь на запит повертається об'єкт JSON, що містить

інформацію про заявку. У цьому об'єкті міститься інформація про валюти, між якими здійснюється обмін, суми обміну, адресу електронної пошти, номер телефону та адресу гаманця для отримання криптовалюти. Крім того, об'єкт містить хеш-код заявки, код заявки та інформацію про те, чи була заявка оброблена.

Зауважте, що сума одержуваних грошей може змінитися при перевірці курсів перед записом. Це пов'язано з тим, що курси валют можуть змінюватися дуже швидко, і щоб надати користувачеві актуальну інформацію, потрібно повторно перевірити курси перед записом. Крім того, повторна перевірка курсів перед записом необхідна для запобігання підробці запиту. Якби не було повторної перевірки, злодій міг би надіслати підроблений запит із зміненою сумою обміну, що призвело б до некоректного обміну валюти. Повторна перевірка курсів перед записом дозволяє уникнути такого підроблення та забезпечує безпеку обміну валют.

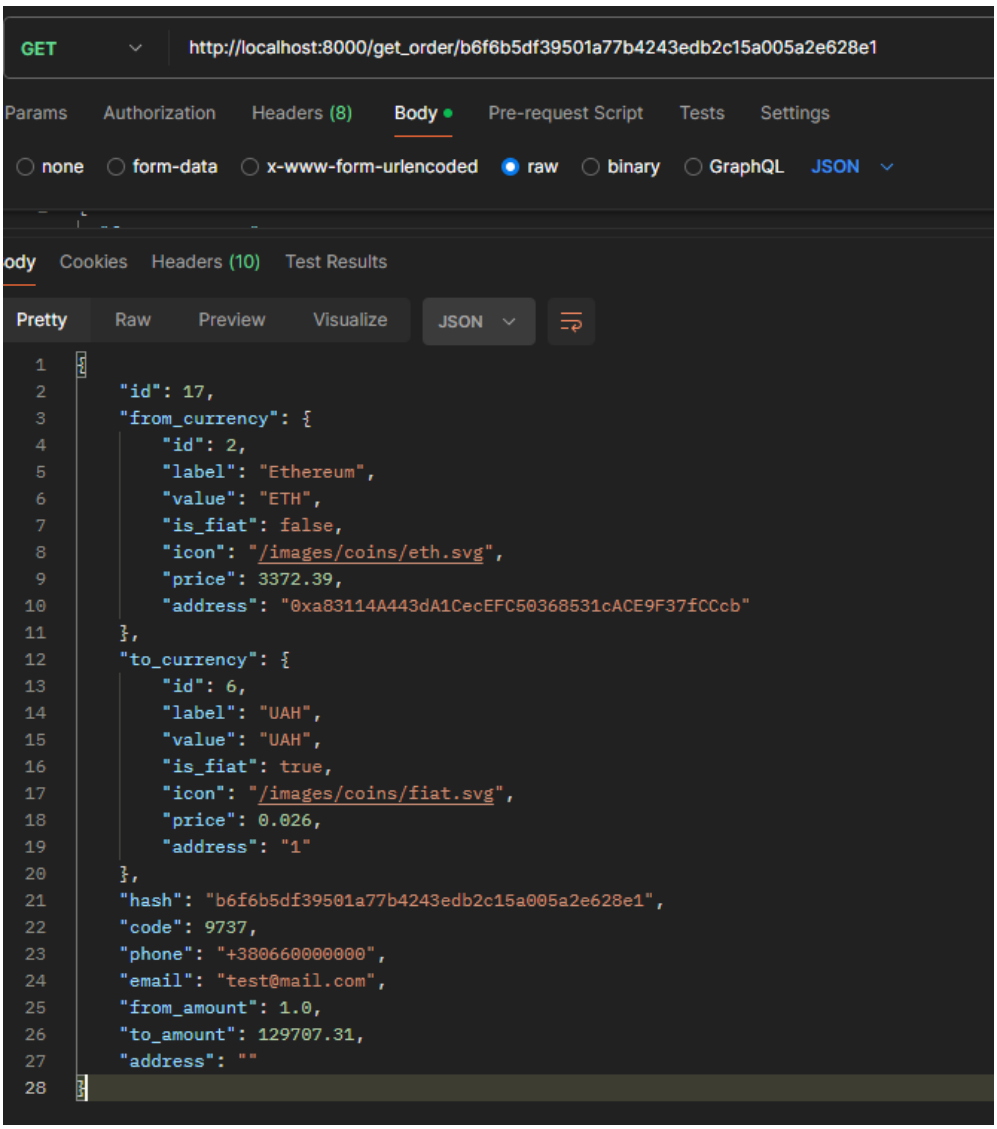

Рис. 2.9. Запит на інформацію про замовлення

2.5.2. Опис клієнтської частини

2.5.2.1. Структура React проекту

Проект має стандартну структуру React (рис. 2.10), яка забезпечує читання, модульність та легкість підтримки коду. Ця структура включає основні папки та файли, кожна з яких має своє призначення та роль у проекті.

Коренева директорія містить ключові файли конфігурації та документації, Папка "public" містить статичні файли, які не піддаються обробці інструментами збирання і безпосередньо доступні з кореневої URL-адреси.

Підпапка '"images" включає різні графічні ресурси, такі як логотипи та іконки, що використовуються в інтерфейсі програми. Підпапка coins містить іконки різних криптовалютів, таких як btc.svg, eth.svg, які використовуються в компонентах інтерфейсу для візуального представлення валют.

Папка "src" є основною для розробки та містить весь вихідний код програми. Основні файли: App.js - кореневий компонент програми, що поєднує всі інші компоненти; index.js - точка входу в програму, де відбувається рендеринг кореневого компонента App в DOM.

Папка "components" містить окремі компоненти програми. Кожен компонент відповідає за певну частину інтерфейсу та інкапсулює свою власну логіку та стилі.

Папка "styles" містить файли CSS, які визначають стилі програми. Основний файл стилів - main.css, який включає глобальні стилі для всієї програми, а також специфічні стилі для окремих компонентів.

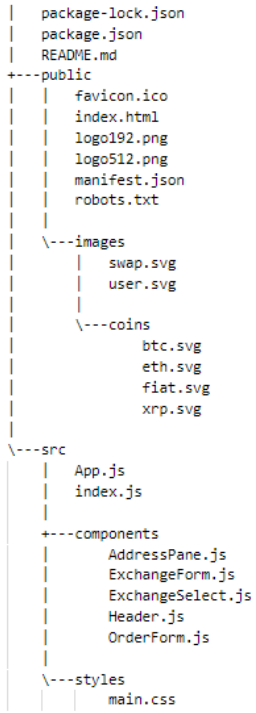

Рис. 2.10. Файлова структура проекту

Використання такої структури у проекті забезпечує кілька ключових переваг: організаційність, модульність, чистоту коду та легкість розширення. Ця структура допомагає забезпечити високу якість розробки та дозволяє ефективно працювати над проектом, підтримувати його та розширювати за необхідності.

2.5.2.2. Взаємодія з сервером

У кореневому компоненті були створені методи для роботи з сервером (рис. 2.11), що значно спростило обмін даними між клієнтською та серверною частинами програми. Ці методи використовували бібліотеку Axios, яка надає зручний інтерфейс для виконання запитів HTTP.

Один із методів був створений для отримання інформації про замовлення. Цей метод відправляв GET-запит на сервер з використанням хешу замовлення як параметр. Якщо сервер успішно обробляв запит і повертав статус 200, дані

замовлення зберігалися в стан програми. У разі помилки викликався спеціальний оброблювач, який керував невдалими запитами.

Інший метод був створенний для отримання списку доступних опцій обміну валют. Він також надсилав GET-запит на сервер і, при успішній відповіді, зберігав отримані дані у стан програми.

Також було створено метод для додавання нового замовлення. Цей метод надсилав POST-запит на сервер із даними нового замовлення. Якщо запит успішно оброблявся і сервер повертав статус 200, автоматично викликався метод отримання інформації про щойно створеному замовленні, використовуючи хеш, повернутий сервером.

```
const getOrder = (order_hash) => {
    axios.get(API_URL + 'get_order/' + order_hash)
        .then(res => {
            res.status === 200 && setOrder(res.data);
        }).catch(error => { faildedRequest() });
};

const getExchangeOptions = () => {
    axios.get(API_URL + 'get_exchange_options')
        .then(res => {
            res.status === 200 && setOptions(res.data);
        }).catch(error => { faildedRequest() });
};

const addOrder = (request) => {
    axios.post(API_URL + 'add_order', request)
        .then(res => {
            res.status === 200 && getOrder(res.data.hash);
        }).catch(error => { faildedRequest() });
}
```

Рис. 2.11. Код з методами взаємодії з сервером

При завантаженні сторінки веб-додатку відразу здійснюється запит до сервера для отримання інформації про доступні курси обміну, списки валют та інші деталі. Ці дані необхідні для створення форми обміну, яка дозволяє користувачам легко вибирати валюти, вказувати суми обміну та надсилати заявки на обмін.

Після отримання даних із сервера, React-компоненти можуть використовувати ці дані для динамічного створення форми обміну. Наприклад, список доступних валют може бути відображений у меню, а інформація про курси обміну може бути використана для надання актуальної інформації про ціни обміну.

Важливо, щоб ця операція відбувалася асинхронно, щоб сторінка не блокувалася під час завантаження даних із сервера. Користувачеві має відображатися повідомлення про завантаження або індикатор завантаження, поки дані ще не отримані.

Такий підхід дозволяє створювати більш динамічні та інтерактивні інтерфейси користувача, які відображають актуальну інформацію і забезпечують плавну роботу.

2.5.2.3. Форма обміну

Форма обміну валют складається з кількох ключових елементів, які полегшують користувачам проведення операцій.

Перший елемент – вибір валюти для обміну. Для цього користувач використовує меню, що випадає, в якому представлені всі доступні валюти.

Другий елемент – введення суми для обміну. Для цього передбачено текстове поле, де користувач зазначає необхідну суму.

Третій елемент – вибір валюти для отримання. Це також випадаюче меню, подібне до попереднього.

Четвертий елемент – введення контактної інформації. Для завершення операції обміну користувач повинен надати свої контактні дані, такі як електронна пошта та номер телефону. Ці дані використовуються для зв'язку з користувачем та підтвердження операції обміну. П'ятий елемент – введення адреси гаманця (за потреби). Якщо користувач обмінює фіатну валюту на

криптовалюту, йому необхідно вказати адресу свого гаманця криптовалютного, на який буде проведений обмін.

Після заповнення всіх необхідних полів користувач може натиснути кнопку обміну, щоб надіслати заявку на обмін (рис. 2.12). Перед відправкою форми здійснюється перевірка на коректність даних. Якщо дані введені некоректно або не відповідають заданим умовам (наприклад, сума обміну знаходиться за межами встановлених лімітів, або введена електронна адреса некоректна), то виводиться відповідне повідомлення про помилку.

```
const onExchangeButtonClick = () => {
    resetErrors();

    const amount = giveInput * giveSelected.price;
    if(isNaN(giveInput) | isNaN(getInput)) {
        setError({...error, general: 'Невірні суми обміну'});
        return;
    } else if(amount > options.limits.max_amount_usd | amount < options.limits.min_amount_usd) {
        setError({...error, general: `Min: ${options.limits.min_amount_usd}$, Max: ${options.limits.max_amount_usd}$`});
        return;
    } else if(email.length < 5 | !email.includes('@')) {
        setError({...error, email: 'Некоректна пошта'});
        return;
    } else if(!isValidPhoneNumber(phone)) {
        setError({...error, phone: 'Некоректний номер'});
        return;
    } else if(giveSelected.is_fiat) {
        if(!isValidCryptoAddress(receiveAddress, getSelected.value)) {
            setError({...error, receiveAddress: 'Некоректний адрес гаманця'});
            return;
        }
    }

    props.onSubmitExchange({
        from_currency: giveSelected.id,
        to_currency: getSelected.id,
        from_amount: giveInput,
        to_amount: getInput,
        email: email,
        phone: phone,
        address: receiveAddress
    });
}
```

Рис. 2.12. Фрагмент коду з перевіркою даних обміну перед відправкою

Для валідації криптовалютних адрес у проекті було використано бібліотеку "multicoin-address-validator". Ця бібліотека надає зручні методи для перевірки валідності криптовалютних адрес різних криптовалют.

Перевага використання цієї бібліотеки полягає в тому, що вона забезпечує перевірку адрес для широкого спектру криптовалют, а також дозволяє легко

інтегрувати перевірку до проекту без необхідності писати власні реалізації валідації для кожної криптовалюти окремо.

Після перевірки коректності всіх даних на клієнтській стороні та їх надсилання на сервер для додаткової перевірки та додавання до бази даних, форма заявки на обмін створюється на клієнтській стороні.

На сервері дані проходять лише етап перевірки та додавання до бази даних. Потім сервер відправляє підтвердження про успішне додавання даних до бази назад на клієнтську сторону.

Після отримання підтвердження про успішне додавання даних до бази клієнта формується форма заявки з урахуванням отриманих даних. Ця форма містить інформацію про замовлення, яку можна використовувати для подальшої обробки замовлення і виконання операції обміну валют.

Для валідації телефонних номерів у проекті використовувалася бібліотека "react-phone-number-input". Ця бібліотека надає компонент введення телефонних номерів, який дозволяє користувачам зручно вводити та перевіряти формат телефонних номерів.

При використанні цієї бібліотеки на стороні форми обміну, користувач може вводити свій номер телефону за допомогою зручного інтерфейсу введення. Бібліотека автоматично перевіряє формат номера, що вводиться, і забезпечує його валідацію, враховуючи формати номерів різних країн.

Це допомагає забезпечити коректне введення номера телефону користувачем, що підвищує зручність використання форми та зменшує ймовірність помилок під час введення контактної інформації.

2.5.2.4. Форма заявок

Форма заявки на обмін валют є інтерфейсом, що відображає інформацію про створену заявку. Вгорі форми відображається номер створеної заявки, що

допомагає ідентифікувати конкретну заявку. Нижче номера заявки відображається унікальний код заявки, який може використовуватися для ідентифікації заявки при зверненні до служби підтримки або її відстеження.

У центральній частині форми відображається інформація про валюти для обміну, суму, яку користувач віддає, та суму, яку він отримує після обміну. Ця інформація представлена у двох колонках, кожна з яких містить зображення валюти та відповідну суму. Залежно від того, чи є валюта, що віддається, криптовалютою або фіатною валютою, відображається відповідна інформація: або адресу для оплати криптовалюти і QR-код, або адресу одержувача фіатної валюти.

Для зменшення навантаження на сервер, генерація QR-коду здійснюється за допомогою стороннього API. Коли клієнт надсилає запит на створення заявки на обмін, разом з іншими даними також надсилається адреса для оплати криптовалюти. Потім на стороні клієнта використовується стороннє API, що спеціалізується на генерації QR-кодів, щоб створити зображення QR-коду на основі отриманої адреси оплати.

У нижній частині форми відображається контактна інформація користувача, включаючи номер телефону та адресу електронної пошти. Ця форма надає користувачеві всю необхідну інформацію про створену заявку на обмін, що дозволяє йому перевірити дані та підтвердити замовлення перед його виконанням.

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Для проекту криптообмінника вхідні та вихідні дані були організовані таким чином:

Вхідні дані:

- Сума обміну: Користувач вводить суму в одній валюті, яку він хоче обміняти.
- Вибрані валюти: Користувач вибирає валюту, яку він хоче обміняти, та валюту, яку він хоче отримати в результаті обміну.
- Контактна інформація користувача: Це може включати адресу електронної пошти, номер телефону та інші деталі, необхідні для зв'язку з користувачем і завершення операції обміну.

Вихідні дані:

- Заявка на обмін Після успішного створення заявки на обмін користувачеві повертається унікальний ідентифікатор заявки (наприклад, номер заявки), який він може використовувати для відстеження статусу свого замовлення.
- Інформація про замовлення: Повертається інформація про обмін, така як сума обміну в обох валютах, курс обміну, контактна інформація користувача та інші деталі замовлення.
- QR-код для оплати: Якщо користувач вибирає криптовалюту для обміну та оплати, йому надається QR-код, який містить адресу для оплати криптовалюти. Це дозволяє користувачеві легко здійснити оплату за допомогою мобільного гаманця.
- Повідомлення про операції: Користувачу надаються повідомлення про успішне створення заявки на обмін, про помилки при введенні даних або про обмеження операції обміну, наприклад, мінімальну або максимальну суму обміну.

Організація вхідних та вихідних даних структурована та зручна для роботи з ними як на стороні клієнта, так і на стороні сервера. Для передачі даних між клієнтом та сервером використовуються стандартні протоколи та формати, такі як JSON для обміну даними та HTTP для передачі запитів та відповідей. Така

організація допомагає забезпечити надійну та ефективну взаємодію між користувачем та додатком.

2.7. Опис розробленого програмного продукту

2.7.1. Використані технічні засоби

Для розробки використовувався персональний комп'ютер із процесором AMD Ryzen 7 3700X, 16 ГБ оперативної пам'яті DDR4 та відеокартою Radeon RX 6800. Для запуску серверної та клієнтської частини достатньо процесора з частотою 2 ГГц та двома ядрами, оперативної пам'яті об'ємом 4 ГБ. Технічні засоби для деплою описані у пункті 1.5.4.

2.7.2. Використані програмні засоби

У проекті криптообмінника використано такі програмні засоби для розробки:

Середовища розробки (IDE):

- PyCharm: Використовується для розробки серверної частини програми на Python. PyCharm надає потужні інструменти для написання, тестування та налагодження коду, такі як підтримка Django, вбудовані тести та відладчик, автодоповнення коду та рефакторинг.
- Visual Studio Code (VS Code): Застосовується для розробки фронтенд частини програми, написаної на JavaScript з використанням React. VS Code популярний завдяки своїм розширенням, таким як підтримка JavaScript/React, інтеграція із системами контролю версій (Git), вбудовані термінали та дебагери.

Інструменти для тестування та налагодження API:

- Postman: Використовується для тестування та налагодження API. За допомогою Postman розробники можуть надсилати HTTP-запити до сервера, перевіряти відповіді, створювати запити для автоматизації тестування та інтеграції з CI/CD пайплайнами.

Засоби для роботи з JavaScript:

- Node.js: Застосовується для роботи з JavaScript на серверній стороні, а також для керування пакетами за допомогою npm (Node Package Manager). Node.js забезпечує виконання серверних скриптів, таких як складання та деплой фронтенд-частини програми.

Фреймворки та бібліотеки:

- Django Rest Framework: Використовується для створення API RESTful на серверній стороні. Django Rest Framework надає потужні інструменти для серіалізації даних, керування запитами та аутентифікації.
- React: Використовується для створення динамічного та чуйного інтерфейсу користувача на клієнтській стороні. React дозволяє будувати компоненти, які оновлюються в реальному часі за зміни стану програми.
- Axios: Застосовується для виконання HTTP-запитів із клієнтської частини програми до сервера. Axios зручний завдяки простому синтаксису та можливості роботи з промісами для обробки асинхронних операцій.

Ці програмні засоби забезпечують зручні та потужні інструменти для створення, тестування та налагодження коду, а також для роботи з API та управління залежностями. Використання цих засобів дозволяє ефективно розробляти та підтримувати проект, забезпечуючи високу якість та надійність кінцевого продукту.

2.7.3. Виклик та завантаження програми

2.7.3.1. Локальний запуск для тестування

Для початку необхідно переконатися, що встановлений Python (рекомендується версія 3.8 або вище) та віртуальне оточення. Створіть віртуальне оточення та активуйте його за допомогою команд "python -m venv venv" та "source venv/bin/activate" (на Windows використовуйте "venv \Scripts\activate"). Потрібно встановити залежність із файлу requirements.txt, виконавши команду "pip install -r requirements.txt".

Застосуйте міграції для налаштування бази даних за допомогою "python manage.py migrate". Створіть суперкористувача для доступу до адміністративної панелі за допомогою команди "python manage.py createsuperuser". Запустіть сервер розробки Django, виконавши команду "python manage.py runserver".

Для запуску веб-додатку необхідно переконатися, що встановлено Node.js (рекомендується версія 14 або вище) та npm. Перейдіть до директорії "client" за допомогою команди "cd client". Потрібно встановити залежність із файлу "package.json", виконавши команду "npm install". Запустіть фронтенд-сервер розробки за допомогою команди "npm start".

2.7.3.2. Розгортання проекту на сервері

Розгортання проекту включає декілька ключових кроків, які забезпечують правильне налаштування як серверної, так і клієнтської частини.

Підготовка до сервера. Для початку необхідно підготувати сервер. Це включає встановлення всіх необхідних програмних пакетів, таких як Python, віртуальне оточення, і системи управління процесами для серверної частини. Також потрібно встановити веб-сервер, який обслуговуватиме клієнтську частину та проксируватиме запити до сервера додатків.

Наступним кроком є клонування проекту на сервер та налаштування віртуального оточення. Після цього потрібно встановити всі залежності, вказані в проекті, щоб забезпечити роботу всіх компонентів програми.

Для цього проекту використовується SQLite, що спрощує процес налаштування бази даних. Необхідно застосувати міграцію бази даних, щоб створити всі необхідні таблиці та структури даних.

Далі потрібно налаштувати серверну програму для роботи в продакшн-режимі. Це включає налаштування сервера додатків, таких як Gunicorn, і створення відповідних конфігураційних файлів для системи управління процесами.

Налаштування клієнтської програми. Для клієнтської частини, написаної на React, необхідно встановити всі залежності та зібрати проект для продакшн-режиму. Зібрані статичні файли використовуватимуться веб-сервером для обслуговування клієнтських запитів.

Налаштування веб-сервера. Веб-сервер, наприклад Nginx, налаштовується обслуговування як статичних файлів, і проксування запитів до сервера додатків. Це забезпечує надійну роботу всього стеку технологій, що використовуються у проекті.

Налаштування статичних та медіа файлів. Проект вимагає окремого налаштування для обробки статичних та медіа файлів. Статичні файли потрібно зібрати та розмістити у відповідній директорії, звідки веб-сервер їх обслуговуватиме.

Конфігурація оточення. Необхідно створити файл конфігурації оточення, який міститиме всі необхідні змінні оточення для коректної роботи програми. Це включає секретні ключі, налаштування бази даних та інші параметри.

Для забезпечення безпеки з'єднань при роботі з веб-програмою критично важливо налаштувати SSL. З'єднання SSL включає кілька ключових кроків.

Перший крок – отримання SSL-сертифіката. Сертифікат можна отримати безкоштовно через Let's Encrypt або придбати у сертифікованого центру залежно від вимог проекту.

Якщо використовується Let's Encrypt, процес можна автоматизувати за допомогою інструменту Certbot, який спрощує отримання та оновлення сертифікатів. Для інших постачальників сертифікатів потрібно виконувати їх інструкції з установки.

Після отримання сертифіката потрібно налаштувати веб-сервер, щоб він використовував SSL для шифрування трафіку. Це включає вказівки шляхів до SSL-сертифіката та приватного ключа у конфігурації веб-сервера. Налаштування SSL вимагає перезапуску веб-сервера для застосування змін.

Ці кроки забезпечують шифрування даних, що передаються між клієнтом та сервером, що захищає конфіденційну інформацію та покращує загальну безпеку веб-програми.

Фінальні кроки. Після виконання всіх вищеописаних кроків слід протестувати розгорнуту систему, щоб переконатися в її правильній роботі. Це включає перевірку доступності всіх сервісів, коректності обробки запитів та роботи всіх функцій програми.

Таким чином, розгортання проекту криптообмінника включає комплексне виконання ряду завдань, спрямованих на налаштування серверного та клієнтського додатків, баз даних та інфраструктури для забезпечення стабільної та безпечної роботи усієї системи.

2.7.4. Опис інтерфейсу користувача

Інтерфейс користувача (UI) веб-додатку розроблений з урахуванням зручності використання та інтуїтивної взаємодії для кінцевого користувача. Було

ретельно опрацьовано кожен елемент дизайну, щоб переконатися, що він відповідає сучасним стандартам та відповідає потребам користувачів.

Весь інтерфейс був виконаний у темних тонах, що забезпечує комфортне використання за будь-якої яскравості екрану. Крім того, використовувалися чіткі та яскраві акценти, щоб виділити ключові елементи та покращити навігацію за додатком. Ця дизайн-концепція також передбачає адаптивність до будь-яких пристроїв, від смартфонів до настільних комп'ютерів, що дозволяє користувачам отримувати доступ до програми у будь-який час та в будь-якому місці.

При переході на сайт користувачеві відразу ж надається вікно з вибором валют для обміну (рис. 2.13). Це вікно містить списки, що випадають, що дозволяють вибрати валюту, яку користувач хоче обміняти і отримати, також їх можна поміняти місцями і тоді відразу ж з'явиться поле введення гаманця (якщо міняється готівка на криптовалюту). Дані про поточні курси валют оновлюються динамічно, надаючи користувачеві актуальну інформацію.

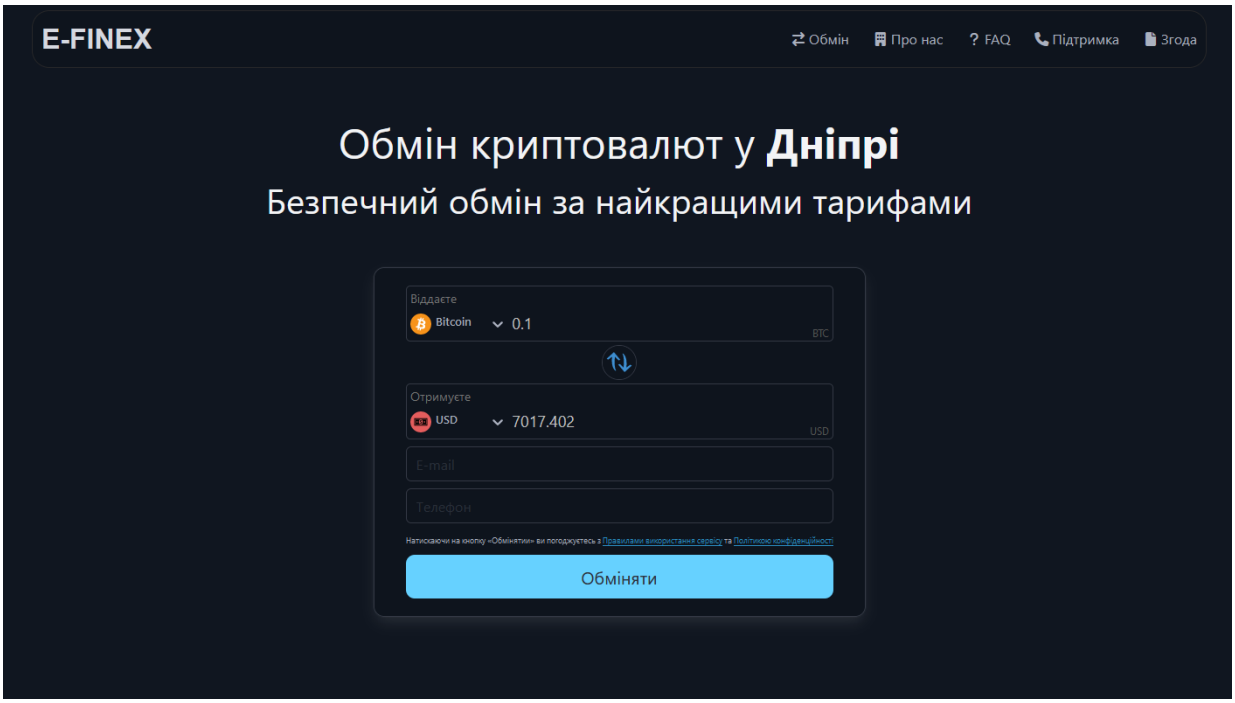

Рис. 2.13. Головна сторінка (форма обміну)

Нижче основної форми користувачі можуть знайти розділи "Про нас" (рис. 2.14) і "FAQ" (рис. 2.15), які надають інформацію про компанію, її місію та відповіді на запитання, що часто задаються. Цей розділ допомагатиме користувачам краще зрозуміти, як працює сервіс і які переваги він пропонує. Також є Google карта з міткою, яка допомагає користувачам знайти фізичне розташування обмінника. На відміну від форми обміну решта інформації статична.

Рис. 2.14. Вкладка з описом

Внизу сторінки також представлені контактні дані (рис. 2.7.3), включаючи адресу електронної пошти, номер телефону та форму зворотного зв'язку, що дозволяє користувачам швидко зв'язатися з підтримкою для отримання додаткової інформації або допомоги.

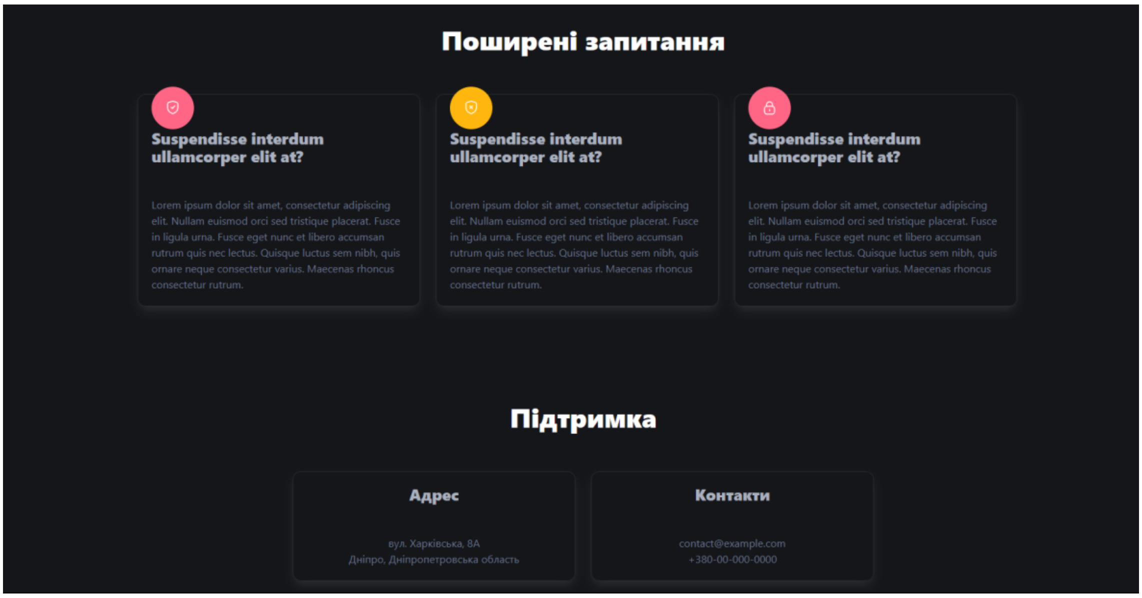

Рис. 2.15. Вкладка з FAQ та контактами

Після успішної перевірки та обробки даних на сервері користувачу надається форма заявки (рис. 2.16). У цій формі відображається інформація про заявку, включаючи номер заявки, код для підтвердження у касира, суму та валюту обміну, а також адресу криптогаманця, якщо це необхідно. Для зручності оплати надається QR-код.

Рис. 2.16. Форма замовлення

В проекті використовується стандартна адміністративна панель Django. Адміністративна панель Django дозволяє адміністраторам та розробникам легко керувати даними та контролювати різні аспекти веб-додатку без необхідності писати додатковий код для створення інтерфейсів керування.

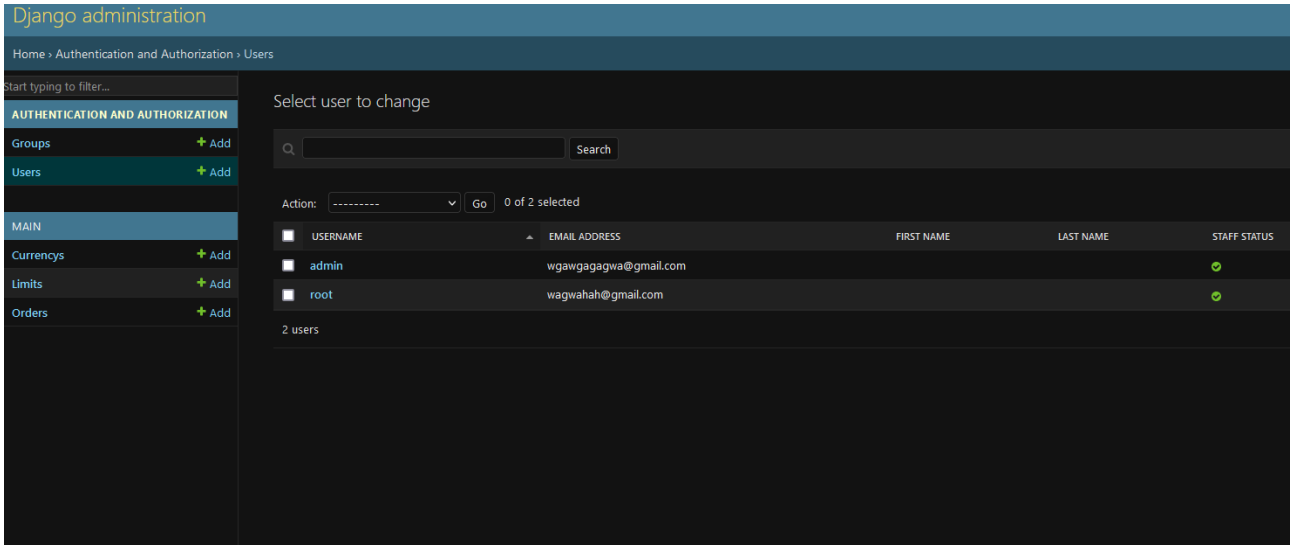

Рис. 2.18. Панель адміністратора

Адміністратори можуть переглядати всі заявки на обмін (рис. 2.18), подані користувачами. Панель надає зручний інтерфейс для фільтрації, сортування та пошуку заявок. Адміністратор може бачити деталі кожної заявки, включаючи інформацію про користувача, тип обміну, суму та статус заявки.

Адміністративна панель дозволяє адміністраторам легко оновлювати курс обміну криптовалют (рис. 2.19). У розділі керування курсами можна додати нові валютні пари. Це забезпечує гнучкість та контроль над процесом обміну, дозволяючи швидко реагувати на зміни ринкових умов.

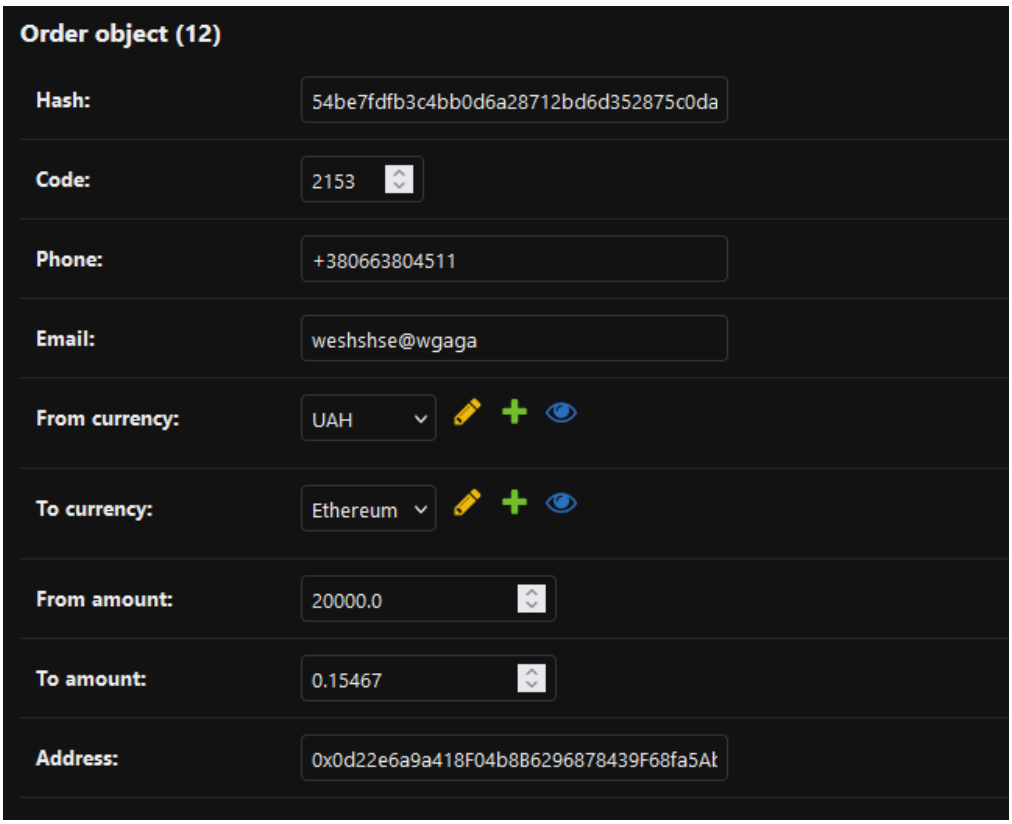

Рис. 2.18. Заявка на обмін з панелі адміністратора

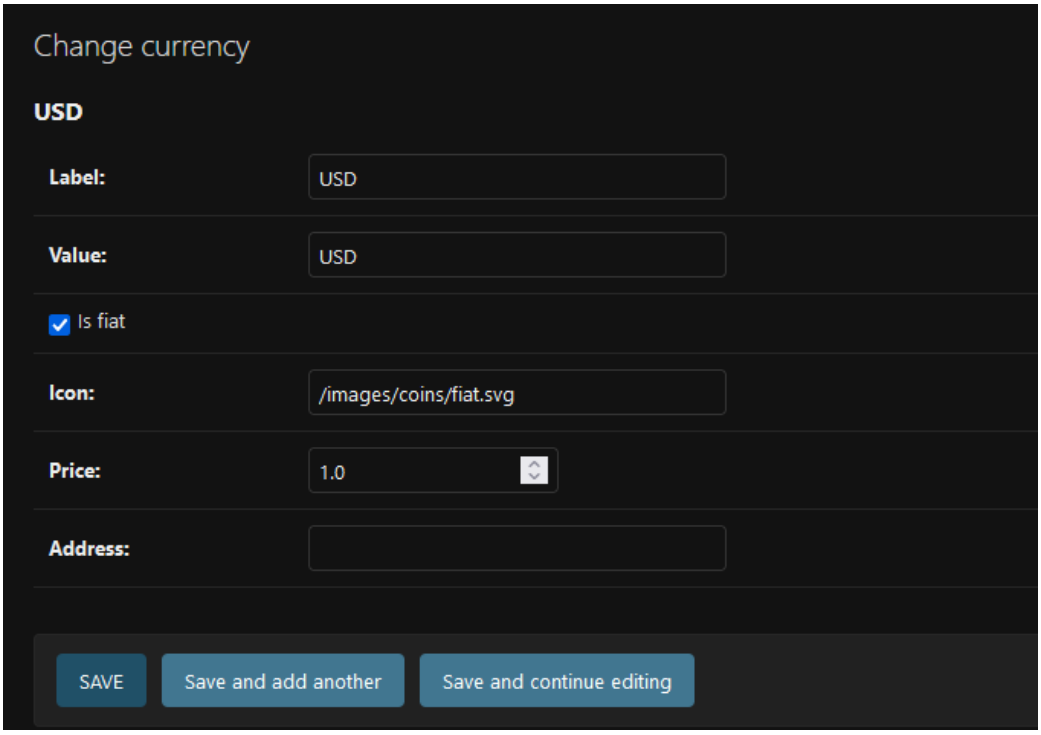

Рис. 2.19. Зміна курсу у панелі адміністратора

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки інформаційної системи

Початкові дані:

1. передбачуване число операторів програми – 824;
2. коефіцієнт складності програми – 1,3;
3. коефіцієнт корекції програми в ході її розробки – 0,07;
4. годинна заробітна плата програміста– 210 грн/год;

Згідно зі статистичними даними з порталу "DOU.UA" за грудень 2023 року, середня зарплата Junior розробника становить 950 доларів на місяць [20]. За поточним курсом 39 грн. за долар це відповідає 37 050 грн. При стандартному робочому графіку о 176 годині на місяць, це виходить близько 210 грн на годину.

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0,8;
7. вартість машино-години ЕОМ – 1,58 грн/год.

При розробці проекту використовувалися ПК та зовнішні пристрої, які сумарно споживають 0,45 кВт за годину. На момент розробки тариф на електроенергію становив 2,64 грн. за кВт·год, що відповідає приблизно 1,18 грн. за годину. Щомісячна оплата за використання інтернету складає 295 грн., що дорівнює приблизно 0,4 грн. в годину. З цього виходить, що сумарна вартість ЕОМ – 1,58 грн./год.

Трудомісткість розробки ПЗ обчислюється за такою формулою:

$$t = t_{\text{о}} + t_{\text{и}} + t_{\text{п}} + t_{\text{отл}} + t_{\text{д}}, \text{ людино-годин,} \quad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (50 людино-годин);

$t_{\text{и}}$ – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

$t_{\text{п}}$ – витрати праці на програмування по готовій блок-схемі;

$t_{\text{отл}}$ – витрати праці на налагодження програми на ЕОМ;

$t_{\text{д}}$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів:

$$Q = q \cdot C \cdot (1 + p), \quad (3.2)$$

де q – передбачуване число операторів (824);

C – коефіцієнт складності програми (1,3);

p – коефіцієнт корекції програми в ході її розробки (0,07);

Розрахунок за формулою (3.2):

$$Q = 824 \cdot 1{,}3 \cdot (1 + 0{,}07) = 1146{,}18$$

Витрати праці на вивчення опису задачі визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75 \ldots 85) \cdot k}, \text{ людино-годин,} \quad (3.3)$$

де Q – умовне число операторів (1146,18),

B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,2),

k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності (1,3).

Розрахунок за формулою (3.3):

$$t_u = \frac{1146{,}18 \cdot 1{,}2}{85 \cdot 0{,}8} = 20{,}22 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20...25)\cdot k}, \text{ людино-годин,} \quad (3.4)$$

Розрахунок за формулою (3.4):

$$t_a = \frac{1146,18}{25 \cdot 0,8} = 57,30 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі, розраховуються за формулою:

$$t_{\text{п}} = \frac{Q}{(20\cdots25)\cdot k}, \text{ людино-годин,} \quad (3.5)$$

Розрахунок за формулою (3.5):

$$t_{\text{п}} = \frac{1146,18}{24\cdot0,8} = 59,69 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4...5)\cdot k}, \text{ людино-годин.} \quad (3.6)$$

Розрахунок за формулою (3.6):

$$t_{\text{отл}} = \frac{1146,18}{5 \cdot 0,8} = 286,54 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{k} = 1,5 \cdot t_{\text{отл}}, \text{ людино-годин.} \quad (3.7)$$

Розрахунок за формулою (3.7):

$$t_{\text{отл}}^{k} = 1,5 \cdot 286,54 = 429,81 \text{ людино-годин.}$$

Трудомісткість підготовки матеріалів і рукопису визначається за формулою:

$$t_{\text{др}} = \frac{Q}{(15...20)\cdot k}, \text{ людино-годин.} \quad (3.8)$$

Розрахунок за формулою (3.8):

$$t_{\text{др}} = \frac{1146{,}18}{20 \cdot 0{,}8} = 71{,}63 \text{ людино-годин.}$$

Трудомісткість редагування, печатки й оформлення документації визначається за формулою:

$$t_{\text{до}} = 0{,}75 \cdot t_{\text{др}}, \text{ людино-годин.} \quad (3.9)$$

Розрахунок за формулою (3.9):

$$t_{\text{до}} = 0{,}75 \cdot 71{,}63 = 53{,}72 \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}}, \text{ людино-годин.} \quad (3.10)$$

Розрахунок за формулою (3.10):

$$t_{\text{д}} = 71{,}63 + 53{,}72 = 125{,}35 \text{ людино-годин.}$$

Розрахунок трудомісткості розробки програмного забезпечення за формулою (3.1):

$$t = 50 + 20{,}22 + 59{,}69 + 429{,}81 + 286{,}54 + 125{,}35 = 971{,}61 \text{ людино-годин.}$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ можна розрахувати за формулою:

$$\text{К}_{\text{ПО}} = \text{З}_{\text{ЗП}} + \text{З}_{\text{Мч}}, \text{ грн.,} \quad (3.11)$$

де $\text{З}_{\text{ЗП}}$ – заробітна плата виконавця програми;

$\text{З}_{\text{Мч}}$ – витрати машинного часу.

Заробітна плата виконавців визначається за формулою:

$$\text{З}_{\text{ЗП}} = t \cdot \text{С}_{\text{ПР}}, \text{ грн.} \quad (3.12)$$

де t – загальна трудомісткість, людино-годин (971,61);

$\text{С}_{\text{ПР}}$ – середня годинна заробітна плата програміста, грн/годину (210).

Розрахунок за формулою (3.12):

$$\text{З}_{\text{ЗП}} = 971{,}61 \cdot 210 = 204038{,}1 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на

ЕОМ, визначається за формулою:

$$З_{Мч} = t_{отл} \cdot С_{Мч}, \text{ грн.}, \quad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год. (286,54);

$С_{Мч}$ – вартість машино-години ЕОМ, грн/год. (1,58).

Розрахунок за формулою (3.13):

$$З_{Мч} = 286{,}54 \cdot 1{,}58 = 452{,}73 \text{ грн.}$$

Розрахунок витрат на створення програмного забезпечення за формулою (3.11):

$$К_{ПО} = 204038{,}1 + 452{,}73 = 204490{,}83 \text{ грн.}$$

Очікуваний час розробки програмного забезпечення розраховується за формулою:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.}, \quad (3.14)$$

де t – загальна трудомісткість, людино-годин (971,61);

B_k – число виконавців (1);

F_p – місячний фонд робочого часу (176 год.).

Розрахунок за формулою (3.14):

$$T = \frac{971{,}61}{1 \cdot 176} = 5{,}52 \text{ міс.}$$

Висновок: Загалом вартість розробки ПЗ виходить 204 490,83 грн., терміном приблизно 5,5 місяців.

ВИСНОВКИ

У рамках кваліфікаційної роботи було успішно спроектовано та розроблено серверну та клієнтську частину криптообмінника, що відповідає всім поставленим умовам та вимогам.

На першому етапі було проведено ретельний аналіз завдання, уточнено всі вимоги та поставлено чітке завдання для подальшої розробки.

Серверна частина криптообмінника була розроблена за допомогою потужного фреймворку Django REST Framework. Було створено надійну базу даних, розроблено моделі, серіалізатори та інтерфейси взаємодії. Також було реалізовано динамічну систему оновлення курсів криптовалют, що дозволяє користувачам отримувати завжди актуальну інформацію. Для перевірки працездатності та надійності серверної частини було проведено базове тестування API.

Клієнтська частина криптообмінника була розроблена за допомогою популярного фреймворку React. Для зручності верстки також був використаний CSS фреймворк Bulma як допоміжний засіб. Було створено зручний та інтуїтивно зрозумілий базовий інтерфейс, розроблено динамічну форму обміну криптовалют. Для взаємодії з сервером API була використана бібліотека Axios.

Користувачу надається можливість:

- отримувати актуальні курси;
- вибирати валюти для обміну;
- подавати заявку на обмін;
- переглядати заявки з їх статусом;
- швидко визначити місцезнаходження обмінника по карті.

У результаті, була успішно розроблена система криптообмінника, що відповідає всім вимогам та забезпечує надійну та зручну роботу для користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Криптовалюта. Wikipedia. URL: https://uk.wikipedia.org/wiki/Криптовалюта (дата звернення: 15.04.2024).
2. Bitcoin. Wikipedia. URL: https://uk.wikipedia.org/wiki/Біткойн (дата звернення: 15.04.2024).
3. SHA-1. Wikipedia. URL: https://uk.wikipedia.org/wiki/SHA-1 (дата звернення: 15.04.2024).
4. SHA-256. Wikipedia. URL: https://uk.wikipedia.org/wiki/SHA-256 (дата звернення: 15.04.2024).
5. Python. Wikipedia. URL: https://uk.wikipedia.org/wiki/Python (дата звернення: 17.04.2024).
6. REST. Wikipedia. URL: https://uk.wikipedia.org/wiki/REST (дата зверненняL: 17.04.2024).
7. Django. Wikipedia. URL: https://uk.wikipedia.org/wiki/Django (дата звернення: 17.04.2024).
8. Django Documentation. Остаточне джерело для Django. URL: https://docs.djangoproject.com/en/stable/ (дата звернення: 20.04.2023).
9. Django Rest Framework. Офіційна документація. URL: https://www.django-rest-framework.org/ (дата звернення: 20.04.2024).
10. W. Richard Stevens. TCP/IP Illustrated, Volume 1: The Protocols. Addison-Wesley Professional, 1994.
11. JavaScript. Wikipedia. URL: https://uk.wikipedia.org/wiki/JavaScript (дата звернення: 20.04.2024)
12. React. Wikipedia. URL: https://uk.wikipedia.org/wiki/React (дата звернення: 20.04.2024)
13. React Documentation. Офіційна документація. URL: https://reactjs.org/docs/getting-started.html (дата звернення: 24.04.2024)

14. SQLite. Wikipedia. URL: https://uk.wikipedia.org/wiki/SQLite (дата звернення: 24.04.2024)
15. Bulma. Офіційний сайт. URL: https://bulma.io/ (дата звернення: 24.04.2024)
16. Серіалізація. Wikipedia. URL: https://uk.wikipedia.org/wiki/Серіалізація (дата звернення: 07.05.2024)
17. J. Duckett. HTML and CSS: Design and Build Websites. John Wiley & Sons, 2011.
18. A. Shenoy. Learning Bulma: Understand How to Develop Responsive, Mobile-first Websites Using This Impressive, Modern Framework. Apress, 2019.
19. Django deployment. Офіційна документація по розгортанню. URL: https://docs.djangoproject.com/en/5.0/howto/deployment/ (дата звернення: 20.05.2024)
20. DOU. Статистичні дані по заробітній платі Junior розробника. URL: https://jobs.dou.ua/salaries/?period=2023-12&position=Junior%20SE (дата звернення: 23.05.2024)

ДОДАТОК А

КОД ПРОГРАМИ

models.py

```
from django.db import models


class Currency(models.Model):
    label = models.CharField(max_length=20)
    value = models.CharField(max_length=5)
    is_fiat = models.BooleanField()
    icon = models.CharField(max_length=100)
    price = models.FloatField()
    address = models.CharField(max_length=256, null=True)

    def __str__(self):
        return self.label


class Order(models.Model):
    hash = models.CharField(max_length=256)
    code = models.IntegerField()
    phone = models.CharField(max_length=16)
    email = models.CharField(max_length=100)
    from_currency = models.ForeignKey(Currency, on_delete=models.CASCADE,
related_name='from_currency')
    to_currency = models.ForeignKey(Currency, on_delete=models.CASCADE,
related_name='to_currency')
```

```
    from_amount = models.FloatField()
    to_amount = models.FloatField()
    address = models.CharField(max_length=256, null=True)

class Limit(models.Model):
    min_amount_usd = models.FloatField()
    max_amount_usd = models.FloatField()
```

serializers.py

```
from rest_framework import serializers
from .models import *
from . import utils

class LimitsSerializer(serializers.ModelSerializer):
    class Meta:
        model = Limit
        fields = ['min_amount_usd', 'max_amount_usd']

class CurrencySerializer(serializers.ModelSerializer):
    class Meta:
        model = Currency
        fields = '__all__'

class OrderCreateSerializer(serializers.ModelSerializer):
    address = serializers.CharField(required=False, allow_blank=True,
allow_null=True)
```

```
    class Meta:
        model = Order
        fields = ['phone', 'email', 'from_currency', 'to_currency', 'from_amount',
'address']


    def create(self, validated_data):
        order_code = utils.get_random_code()
        order_hash = utils.get_hash(validated_data['phone'], order_code)
        order_amount = (validated_data['from_amount'] *
                        validated_data['from_currency'].price /
validated_data['to_currency'].price)


        decimal_places = 5
        if validated_data['to_currency'].is_fiat:
            decimal_places = 2


        order = Order.objects.create(
            hash=order_hash,
            code=order_code,
            to_amount=round(order_amount, decimal_places),
            **validated_data
        )


        return order


class OrderReadSerializer(serializers.ModelSerializer):
    from_currency = CurrencySerializer()
```

```
    to_currency = CurrencySerializer()

    class Meta:
        model = Order
        fields = '__all__'
```

prices.py

```
from django.utils import timezone
from .models import Currency
import requests
import json

last_update = None

def get_prices(symbols):
    for x in range(len(symbols)):
        symbols[x] = symbols[x] + 'USDT'

    api_request = ('https://api.binance.com/api/v3/ticker/price?symbols=' +
                   json.dumps([str(x) for x in symbols]).replace(' ', ''))
    response = requests.get(api_request)
    if response.status_code == 200:
        return json.loads(response.text)
    return None



def update():
```

```
    global last_update
    if last_update is not None:
        current_time = timezone.now()
        time_difference = current_time - last_update
        minutes = time_difference.total_seconds() // 60

        if minutes < 1:
            return

    coins = Currency.objects.filter(is_fiat=False)
    symbols = [record.value for record in coins]

    prices = get_prices(symbols)

    if prices is not None:
        for x in prices:
            symbol = x['symbol'].replace('USDT', '')
            coin = Currency.objects.get(value=symbol)

            coin.price = float(x['price'])
            coin.save()

        last_update = timezone.now()
```

urls.py

```
from django.urls import path
from . import views
```

```
urlpatterns = [
    path('get_exchange_options', views.get_exchange_options, name='get-exchange-options'),
    path('add_order', views.add_order, name='add-order'),
    path('get_order/<str:order_hash>', views.get_order, name='get-order')
]
```

views.py

```
from rest_framework.response import Response
from rest_framework.decorators import api_view
from rest_framework import status
from .serializers import *
from . import prices

@api_view(['GET'])
def get_exchange_options(request):
    prices.update()

    limits = Limit.objects.last()
    crypto_currencies = Currency.objects.filter(is_fiat=False)
    fiat_currencies = Currency.objects.filter(is_fiat=True)
    response_data = {
        "limits": LimitsSerializer(limits).data,
        "give": CurrencySerializer(crypto_currencies, many=True).data,
        "get": CurrencySerializer(fiat_currencies, many=True).data
    }
```

```
    return Response(response_data, status=status.HTTP_200_OK)


@api_view(['POST'])
def add_order(request):
    data = request.data
    serializer = OrderCreateSerializer(data=data)
    if serializer.is_valid():
        order = serializer.save()
        return Response({
            'hash': order.hash
        }, status=status.HTTP_200_OK)
    return Response(serializer.errors, status=status.HTTP_200_OK)


@api_view(['GET'])
def get_order(request, order_hash):
    order = Order.objects.filter(hash=order_hash)
    if order.exists():
        return Response(OrderReadSerializer(order.last()).data, status.HTTP_200_OK)


    return Response(status=status.HTTP_404_NOT_FOUND)
```

App.js

```
import React, {useState, useEffect} from 'react';
import Header from './componenets/Header';
import ExchangeForm from './componenets/ExchangeForm';
import OrderForm from './componenets/OrderForm';
```

```
import axios from 'axios';

const API_URL = "http://localhost:8000/";

function App() {
    const [order, setOrder] = useState(null);
    const [options, setOptions] = useState(null);
    const [error, setError] = useState('');

    useEffect(() => {
        const search = window.location.search;
        const params = new URLSearchParams(search);
        const order_hash = params.get('order');

        getExchangeOptions();
        order_hash !== null && getOrder(order_hash);
    }, []);

    const onSubmitExchange = (request) => {
        addOrder(request);
    };

    const getOrder = (order_hash) => {
        axios.get(API_URL + 'get_order/' + order_hash)
            .then(res => {
                res.status === 200 && setOrder(res.data);
            }).catch(error => { faildedRequest() });
```

```
};

const getExchangeOptions = () => {
    axios.get(API_URL + 'get_exchange_options')
        .then(res => {
            res.status === 200 && setOptions(res.data);
        }).catch(error => { faildedRequest() });
};

const addOrder = (request) => {
    axios.post(API_URL + 'add_order', request)
        .then(res => {
            res.status === 200 && getOrder(res.data.hash);
        }).catch(error => { faildedRequest() });
}

const faildedRequest = () => {
    setError('Помилка при надсиланні запиту, спробуйте пізніше.');
}

return(
<section className='hero h-dark-background is-fullheight'>
    <Header />
    <div className="container mt-6 mb-6">
        <div className="has-text-centered has-text-light">
            <h1 className="is-size-1 is-size-4-mobile">Обмін криптовалют у
<span><b>Дніпрі</b></span></h1>
```

```
                <h3 className="is-size-2 is-size-5-mobile">Безпечний обмін за найкращими тарифами</h3>
            </div>
            <small className='has-text-danger'>{error}</small>
            { order !== null ? <OrderForm order={order} /> : (order === null & options !== null && <ExchangeForm onSubmitExchange={onSubmitExchange} options={options} setOptions={setOptions} /> )}

        </div>
    </section>
    );
}
export default App;
```

OrderForm.js

```
import React from "react";
import AddressPane from "./AddressPane";

function OrderForm(props) {
    const order = props.order;

    return (
        <div className="container my-6">
            <div className="container order-form">
                <div className="box mt-6 p-dark-background border-color-def">

                    <div className="has-text-centered bottom-border-def py-2 is-mobile">
```

```
                <b className="has-text-grey-light is-size-3">Заявка
№{order.id}</b>
            </div>

            <div className="has-text-centered">
                <small className="has-text-grey-light">Код: <span
className="has-text-danger">{order.code}</span></small>
            </div>

            <div className="mx-6 mt-5">

                <div className="container">
                    <div className="columns is-centered is-gapless">

                        <div className="column is-5 is-half border-color-def border-
radius-def">
                            <div className="columns my-1 is-mobile is-gapless is-
vcentered is-centered">

                                <div className="column is-flex is-justify-content-right">
                                    <figure className="image is-48x48">
                                        <img src={order.from_currency.icon} alt="" />
                                    </figure>
                                </div>

                                <div className="column ml-1">
                                    <p className="is-size-6 is-size-7-mobile has-text-
grey">Віддаєте</p>
```

```
                                <small className="is-size-7 has-text-grey-
light"><b>{order.from_amount} {order.from_currency.value}</b></small>
                            </div>


                        </div>
                    </div>

                    <div className="column is-2 has-text-centered">
                        <figure className="image is-48x48 is-inline-block">
                            <img src="/images/h-swap.svg" alt="" />
                        </figure>
                    </div>

                    <div className="column is-5 is-half border-color-def border-
radius-def">
                        <div className="columns my-1 is-mobile is-gapless is-
vcentered is-centered">

                            <div className="column is-flex is-justify-content-right">
                                <figure className="image is-48x48">
                                    <img src={order.to_currency.icon} alt="" />
                                </figure>
                            </div>

                            <div className="column ml-1">
                                <p className="is-size-6 is-size-7-mobile has-text-
grey">Отримуєте</p>
```

<small className="is-size-7 has-text-grey-light">{order.to_amount} {order.to_currency.value}</small>

{

!order.from_currency.is_fiat ? (

<AddressPane title="Адрес для оплати:"

address={order.from_currency.address}

image={`https://api.qrserver.com/v1/create-qr-code/?size=120x120&data=${order.from_currency.address}`}

prompt={`Переведіть ${order.from_amount} ${order.from_currency.value} на цей гаманець, після оплати повідомте касиру номер вашої заявки та код.`}

/>

) : (

<AddressPane title="Адрес отримувача:"

address={order.address}

image="/images/user.svg"

prompt={`Повідомте касиру номер вашої заявки або номер телефону, після оплати ${order.to_amount} ${order.to_currency.value} буде відправлено на цей гаманець.`}

/>

```
                    )
                }

                <div className="columns mt-3 is-centered is-mobile">
                    <div className="column">
                        <div className="has-text-left"><small className="is-size-7
dark-text">{order.phone}</small></div>
                    </div>
                    <div className="column">
                        <div className="has-text-right"><small className="is-size-7
dark-text">{order.email}</small></div>
                    </div>
                </div>


            </div>
        </div>
    </div>
</div>
);
}
export default OrderForm;
```

ExchangeForm.js

```
import React, {useState, useEffect, useRef} from 'react';
import ExchangeSelect from './ExchangeSelect';

import { isValidPhoneNumber } from 'react-phone-number-input';
```

```
import PhoneInput from 'react-phone-number-input/input';
import {validate as isValidCryptoAddress} from 'multicoin-address-validator';

function ExchangeForm(props) {

    const options = props.options;
    const setOptions = props.setOptions;

    const [giveInput, setGiveInput] = useState(0.1);
    const [getInput, setGetInput] = useState('');
    const [giveSelected, setGiveSelected] = useState(options.give[0]);
    const [getSelected, setGetSelected] = useState(options.get[0]);

    const [phone, setPhone] = useState('');
    const [email, setEmail] = useState('');
    const [receiveAddress, setReceiveAddress] = useState('');

    const [error, setError] = useState({
        email: '',
        phone: '',
        receiveAddress: '',
        general: ''
    });

    const getInputRef = useRef(null);
```

```
  const resetErrors = () => { setError({email: '', phone: '', general: '', receiveAddress: ''}) };

  const onChangeInput = (id, value) => {
    id === 0 ? setGiveInput(value) : setGetInput(value);
    id !== 0 && (getInputRef.current = true);
  }

  const onChangeSelect = (id, option) => {
    id === 0 ? setGiveSelected(option) : setGetSelected(option);
  }

  useEffect(() => {
    setGetInput(giveInput * giveSelected.price / getSelected.price);
  }, [giveInput, giveSelected, getSelected]);

  useEffect(() => {
    if (getInputRef.current) {
      setGiveInput(getInput * getSelected.price / giveSelected.price);
      getInputRef.current = false;
    }
  }, [getInput]);

  useEffect(() => {
    resetErrors();
  }, [phone, email, getInput, giveInput, receiveAddress]);
```

```
  const onExchangeButtonClick = () => {
    resetErrors();

    const amount = giveInput * giveSelected.price;
    if(isNaN(giveInput) | isNaN(getInput)) {
      setError({...error, general: 'Невірні суми обміну'});
      return;
    } else if(amount > options.limits.max_amount_usd | amount < options.limits.min_amount_usd) {
      setError({...error, general: `Min: ${options.limits.min_amount_usd}$, Max: ${options.limits.max_amount_usd}$`});
      return;
    } else if(email.length < 5 | !email.includes('@')) {
      setError({...error, email: 'Некоректна пошта'});
      return;
    } else if(!isValidPhoneNumber(phone)) {
      setError({...error, phone: 'Некоректний номер'});
      return;
    } else if(giveSelected.is_fiat) {
      if(!isValidCryptoAddress(receiveAddress, getSelected.value)) {
        setError({...error, receiveAddress: 'Некоректний адрес гаманця'});
        return;
      }
    }

    props.onSubmitExchange({
      from_currency: giveSelected.id,
```

```
        to_currency: getSelected.id,
        from_amount: giveInput,
        to_amount: getInput,
        email: email,
        phone: phone,
        address: receiveAddress
      });
    }

    return (
      <div className="container is-flex is-justify-content-center">
        <div className="box mt-6 p-dark-background" style={{border: '1px solid #353a46'}}>
          <div className="mr-4 ml-4">
            <small class='is-size-7 has-text-danger'>{error.general}</small>

            <ExchangeSelect options={options.give} id={0} onChangeInput={onChangeInput} inputValue={giveInput} onChangeSelect={onChangeSelect} selectedOption={giveSelected} />

            <div className="container is-flex is-justify-content-center mt-1 mb-1">
              <img src="/images/swap.svg" width="40" alt="" className="rounded-image clickable" onClick={() =>  {
                setOptions(prevState => ({ get: prevState.give, give: prevState.get, limits: prevState.limits }));
                setGiveSelected(getSelected);
                setGetSelected(giveSelected);
              }}/>
```

```
            </div>

            <ExchangeSelect options={options.get} id={1}
onChangeInput={onChangeInput} inputValue={getInput}
onChangeSelect={onChangeSelect} selectedOption={getSelected} />

            <div className="container mt-2 mb-1">

              <div className="mt-2">
                <small class='is-size-7 has-text-danger'>{error.email}</small>
                <input type="text" className="input p-dark-background border-
color ph-color" placeholder="E-mail" onChange={(event) => {
                  setEmail(event.target.value);
                }} />
              </div>

              <div className="mt-2">
                <small class='is-size-7 has-text-danger'>{error.phone}</small>
                <PhoneInput
                  placeholder="Телефон"
                  country='UA'
                  className={`input p-dark-background border-color ph-color
${error.phone.length > 0 && 'border-color-danger'}`}
                  onChange={setPhone}/>
              </div>

              {giveSelected.is_fiat && (
                <div className="mt-2">
```

```
                    <small class='is-size-7 has-text-
danger'>{error.receiveAddress}</small>
                    <input type="text" className="input p-dark-background border-
color ph-color" placeholder={`Ваш ${getSelected.value} адрес`}
onChange={(event) => {
                        setReceiveAddress(event.target.value);
                    }} />
                </div>
            )}

        </div>

        <small style={{fontSize: 'xx-small'}}>Натискаючи на кнопку
«Обмінятии» ви погоджуєтесь з <a href="#" className="is-link">Правилами
використання сервісу</a> та <a href="#" className="is-link">Політикою
конфіденційності</a></small>
        <div className="container mt-2">
            <button className="button is-medium is-fullwidth is-info has-text-
dark" onClick={onExchangeButtonClick}>Обміняти</button>
        </div>
      </div>
    </div>
  </div>
  );
}
export default ExchangeForm;
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	**Опис**
Пояснювальні документи	
ПЗ_Сідоров.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
ПЗ_Сідоров.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Програма	
ExchangeSrc.zip	Архів. Містить коди програми.
Презентація	
Презентація_Сідоров.ppt	Презентація кваліфікаційної роботи.