МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ РОБОТИ З МНЕМОНІЧНИМИ ФРАЗАМИ»**

Виконав: студент 4 курсу, групи 6.1219-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
А.О. Хліпітько
(ініціали та прізвище)
Керівник старший викладач кафедри програмної інженерії, PhD, Чопорова О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

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

_____ Лісняк А.О.
(підпис)

“ 07 ” 02 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Хліпітьку Артему Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка програмного забезпечення для роботи з мнемонічними фразами

керівник роботи Чопорова Оксана Володимирівна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Програмне забезпечення для роботи з мнемонічними фразами.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 07.02.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.02.2023	
2.	Збір вихідних даних.	20.02.2023	
3.	Обробка методичних та теоретичних	06.03.2023	
	джерел.		
4.	Розробка першого та другого розділу.	03.04.2023	
5.	Розробка третього розділу.	17.05.2023	
6.	Оформлення та нормоконтроль	01.06.2023	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	23.06.2023	

Студент _____ А.О. Хліпітько
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.В. Чопорова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка програмного забезпечення для роботи з мнемонічними фразами»: 50 с., 15 рис., 13 джерел.

ГРАФІЧНИЙ ІНТЕРФЕЙС, КРИПТОГАМАНЕЦЬ, МНЕМОНІЧНІ ФРАЗИ, ПЕРЕВІРКА БАЛАНСУ, РОЗРОБКА ДОДАТКУ.

Об’єкт дослідження – розробка програмного забезпечення для роботи з мнемонічними фразами.

Мета роботи: розробити додаток з інтуїтивно зрозумілим графічним інтерфейсом, який дозволить користувачам зручно керувати своїми криптогаманцями.

Метод дослідження – аналітичний.

В роботі розглянуто розробку додатку з графічним інтерфейсом для роботи з мнемонічними фразами. Описано функціонал програми, який дозволяє забезпечити автоматизовану масову генерацію гаманців та перевірку їх балансу шляхом підключення до блокчейн мереж. Розроблений додаток може бути впроваджений у різних сферах, включаючи криптовалютні платформи, фінансові установи та дослідницькі проєкти. Використання програми дозволить автоматизувати та спростити процеси генерації гаманців та перевірки балансу, забезпечуючи швидкість, точність та ефективність у роботі з криптогаманцями.

SUMMARY

Bachelor’s qualifying paper «Development of a Software for Working with Mnemonic Phrases»: 50 pages, 15 figures, 13 references.

GUI, CRYPTO WALLET, MNEMONIC PHRASES, BALANCE CHECK, APPLICATION DEVELOPMENT.

The object of the study is development of an application with a graphical interface for working with a Mnemonic Phrases.

The aim of the study is development an application with an intuitive graphical interface that allows users to conveniently manage their cryptowallets.

The method of research is analytical.

The paper explores the development of an application with a graphical interface for working with mnemonic phrases. The functionality of the program is described, which enables automated mass generation of wallets and checking their balance by connecting to blockchain networks. The developed application can be implemented in various fields, including cryptocurrency platforms, financial institutions, and research projects. The use of the program will automate and simplify the processes of wallet generation and balance checking, ensuring speed, accuracy, and efficiency in working with cryptowallets.

ЗМІСТ

ВСТУП

Дана кваліфікаційна робота присвячена розробці програмного забезпечення для роботи з мнемонічними фразами. Мнемонічні фрази, відомі також як seed phrases, є важливим елементом в криптографії і забезпечують доступ до криптогаманців та цифрових активів. З ростом популярності криптовалют та необхідності забезпечення безпеки цих активів, розробка ефективного програмного забезпечення для роботи з мнемонічними фразами є актуальною та значущою.

Метою даної роботи є розробка програмного забезпечення з інтуїтивно зрозумілим і легким у використанні інтерфейсом, яке дозволить користувачам зручно керувати своїми мнемонічними фразами. Для досягнення цієї мети, в роботі розглядаються основні аспекти роботи з мнемонічними фразами, такі як створення, збереження, відновлення та використання цих фраз.

Завдання, які будуть виконуватись у рамках роботи, включають:

- аналіз існуючих методів та підходів до роботи з мнемонічними фразами;
- розробка програмного забезпечення з інтуїтивно зрозумілим графічним інтерфейсом;
- реалізація функціоналу, що дозволяє створювати, зберігати, та використовувати мнемонічні фрази;
- проведення тестування програмного забезпечення для перевірки його функціональності та ефективності.

Об’єктом дослідження є процес роботи з мнемонічними фразами, а предметом дослідження є розробка програмного забезпечення для полегшення цього процесу.

Отримані результати мають теоретичне значення, оскільки сприяють поглибленню розуміння процесу роботи з мнемонічними фразами та виявленню оптимальних підходів до їх реалізації у програмному забезпеченні. Крім того, розроблене програмне забезпечення має практичне значення, оскільки може бути

використане криптовалютними платформами, фінансовими установами та користувачами для зручного управління мнемонічними фразами.

Отримані результати мають також новизну, оскільки розроблене програмне забезпечення пропонує новий підхід до роботи з мнемонічними фразами, забезпечуючи зручний та легкий у використанні інтерфейс для користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд понять і термінології

Мнемонічна фраза (Mnemonic Phrase) – мнемонічна фраза, також відома як seed phrase або recovery phrase, є послідовністю слів, які можуть використовуватися для відновлення криптогаманця або створення нового гаманця. Вона є легким для запам'ятовування способом збереження приватного ключа і забезпечує доступ до криптовалюти.

Криптогаманець (Crypto Wallet) – криптогаманець є програмним або апаратним засобом, який забезпечує зберігання, управління і використання криптовалюти. Він містить публічні та приватні ключі, необхідні для взаємодії з блокчейном і здійснення транзакцій [1].

Криптовалюта (Cryptocurrency) – криптовалюта є цифровою або віртуальною валютою, яка використовується як засіб обміну і засіб збереження вартості. Вона базується на криптографічних принципах для забезпечення безпеки та контролю над транзакціями.

Ключі (Keys) – ключі використовуються для криптографічного шифрування, розшифрування, підпису або перевірки транзакцій. У контексті роботи з мнемонічними фразами, приватні ключі використовуються для контролю над криптовалютою та доступу до кошельків.

Відновлення гаманця (Wallet Recovery) – це процес відновлення доступу до гаманця, який був втрачений або пошкоджений. Часто використовується за допомогою мнемонічних фраз, які вводяться для відновлення приватних ключів і відповідного доступу до криптовалюти.

Гаманець з одноразовим використанням (One-Time Wallet) – це тип гаманця, який генерується для одноразових транзакцій або короткострокового використання. Він надає тимчасовий доступ до криптовалюти, після чого втрачає свою дійсність.

Приватний ключ (Private Key) – приватний ключ є конфіденційним, секретним числовим значенням, яке використовується для розшифрування, підпису або доступу до криптографічного гаманця. Він є важливим елементом безпеки, оскільки знання приватного ключа дозволяє контролювати власність над криптовалютою, здійснювати транзакції та взаємодіяти з блокчейном.

Публічна адреса (Public Address) – публічна адреса є унікальним ідентифікатором, який використовується для отримання криптовалюти від інших користувачів. Вона генерується на основі приватного ключа та використовується для відображення доступних коштів на блокчейні. Публічна адреса може бути розподілена іншим користувачам для здійснення транзакцій або отримання платежів [2].

Цей огляд понять і термінології надасть основні терміни, які будуть використовуватися в дослідженні і допоможе читачеві краще зрозуміти далі викладений матеріал.

1.2 Аналіз існуючих методів роботи з мнемонічними фразами

Існує велика кількість різних підходів і методів, що використовуються для створення, збереження, відновлення та використання мнемонічних фраз. Нижче приведені деякі методології та їхні переваги та недоліки.

1.2.1 Стандарт BIP-39

Стандарт BIP-39 є важливим компонентом в криптовалютній галузі, оскільки він встановлює протокол для генерації мнемонічних фраз, які використовуються для створення та відновлення гаманців.

Однією з основних переваг стандарту BIP-39 є його простота використання. Мнемонічні фрази складаються з випадкового набору слів, які

зручно запам'ятовувати і вводити на пристроях з обмеженим інтерфейсом, таких як мобільні телефони. Крім того, такі фрази можуть бути легко записані на папері або збережені у формі електронного документу.

Ще одна перевага стандарту BIP-39 полягає в його широкій підтримці у багатьох криптовалютних гаманцях. Багато програмних гаманців та апаратних пристроїв для зберігання криптовалют підтримують стандарт BIP-39, що дозволяє використовувати одну і ту саму мнемонічну фразу для створення і відновлення гаманців різних криптовалют.

Проте, стандарт BIP-39 також має свої можливі недоліки. Одним з них є залежність від набору слів. У стандарті використовується списки слів, які вибираються зі словника. Це означає, що правильний порядок та правильна кількість слів у мнемонічній фразі є критичними для відновлення гаманця. Якщо слова вводяться неправильно або у неправильному порядку, це може призвести до втрати доступу до кошелька та криптовалюти [4].

Також існує ризик втрати мнемонічних фраз. Якщо мнемонічна фраза була записана на папері і цей папір загублено або пошкоджено, то доступ до гаманця може бути втрачений назавжди. Це підкреслює важливість правильного зберігання мнемонічної фрази, можливо, шляхом резервного копіювання у безпечне місце або використання апаратних гаманців, які автоматично резервують мнемонічну фразу.

Усі ці фактори повинні бути враховані користувачами криптовалют при використанні стандарту BIP-39. Важливо бути уважними та відповідальними при створенні, зберіганні і відновленні гаманців, щоб уникнути втрати коштів.

1.2.2 Hierarchical Deterministic Wallets (HD гаманці)

HD (ієрархічно-детерміновані) гаманці використовують ієрархічну структуру для створення множини ключів, що базуються на одній мнемонічній фразі або вихідному "seed" значенні. Цей підхід дозволяє створювати безліч

приватних ключів та відповідних публічних адрес з однієї мнемонічної фрази, що забезпечує зручність та безпеку при керуванні криптовалютними активами.

Однією з головних переваг HD гаманців є легкість резервного копіювання. Замість необхідності резервного копіювання кожного приватного ключа окремо, достатньо зберегти лише мнемонічну фразу або seed значення. Другою важливою перевагою HD гаманців є можливість створювати безліч підгаманців з однієї мнемонічної фрази. Це дозволяє організувати криптовалютні активи в різні групи або підгаманці для кращого управління та приватності.

Проте, з великою кількістю підгаманців можуть виникати певні проблеми з управлінням багатьма ключами. Кожен підгаманець має свій власний приватний ключ та публічну адресу, що вимагає збереження та управління всіма цими ключами. При великій кількості підгаманців може виникнути плутанина та складнощі при організації та резервному копіюванні всіх приватних ключів.

Загалом, HD гаманці забезпечують зручність та безпеку при створенні та управлінні багатьма ключами на основі однієї мнемонічної фрази. Вони є популярними серед криптовалютних користувачів та дозволяють зберігати й управляти криптовалютними активами ефективно та надійно.

1.2.3 Multisig гаманці

Multisig (багато підписів) гаманці дозволяють використовувати кілька підписів для авторизації транзакцій. Вони забезпечують більшу безпеку, оскільки для здійснення транзакцій потрібні підписи декількох учасників. Аналізуються переваги таких гаманців, зокрема забезпечення захисту від втрати чи крадіжки приватного ключа, але також розглядаються потенційні проблеми та обмеження, пов'язані зі складністю використання та необхідністю співпраці між учасниками [3].

Аналізуючи існуючі методи роботи з мнемонічними фразами, розглядаються їхні переваги, недоліки та обмеження. Цей аналіз допомагає

зрозуміти, які методи можуть бути найбільш ефективними та безпечними в конкретному контексті розробки програмного забезпечення для роботи з мнемонічними фразами.

1.3 Аналіз практичного використання мнемонічних фраз

Практичне використання мнемонічних фраз можна проаналізувати у різних сферах, включаючи криптовалютні платформи, фінансові установи та дослідницькі проєкти. Нижче розглянуто основні використання мнемонічних фраз у кожній з цих сфер, а також проаналізовані виклики та проблеми, що виникають у процесі їх застосування.

1.3.1 Криптовалютні платформи

Мнемонічні фрази широко використовуються в криптовалютних гаманцях та платформах обміну криптовалютами. Головні переваги використання мнемонічних фраз полягають у зручності збереження та відновлення гаманців. Користувачі можуть створити мнемонічну фразу, яка є запам'ятовувальним словосполученням, що відповідає приватному ключу гаманця. Це спрощує процес створення резервних копій і відновлення доступу до гаманця у разі втрати приватного ключа.

Однак, використання мнемонічних фраз також створює проблеми безпеки. Неправильне збереження або викрадення мнемонічних фраз може призвести до неправомірного доступу до гаманця та втрати криптовалютних активів. Безпека мнемонічних фраз стає особливо важливою у випадку криптовалют, оскільки їхність цифрові активи, які можуть бути легко передані та використані зловмисниками.

Шляхи подолання цих викликів полягають у застосуванні належних

заходів безпеки. Користувачам рекомендується зберігати мнемонічні фрази у безпечних місцях. Використання криптографічних методів шифрування та багаторівневої аутентифікації може значно підвищити безпеку мнемонічних фраз.

Інноваційні підходи до використання мнемонічних фраз у криптовалютних платформах включають розвиток біометричних методів аутентифікації, таких як використання відбитків пальців або сканування обличчя для доступу до гаманця. Такі підходи можуть покращити безпеку та зручність використання мнемонічних фраз у криптовалютних середовищах.

1.3.2 Фінансові установи

Мнемонічні фрази також застосовуються в фінансових установах, наприклад, при створенні банківських гаманців або облікових записів. Вони можуть спрощувати процес реєстрації та використання фінансових послуг, забезпечуючи зручний спосіб доступу до особистих фінансових ресурсів.

Однак, існують виклики та проблеми, пов'язані з використанням мнемонічних фраз у фінансових установах. Ризик втрати або неправильного використання мнемонічних фраз може призвести до небажаного доступу до фінансових активів або порушення конфіденційності. Крім того, можливість зламування мнемонічних фраз шляхом підбору або атаки може становити серйозну загрозу для безпеки фінансових установ.

Для подолання цих викликів фінансові установи можуть використовувати розширені методи аутентифікації, такі як двофакторна аутентифікація або біометричні методи, для забезпечення безпеки при використанні мнемонічних фраз. Також важливо забезпечити належне навчання клієнтів щодо безпеки мнемонічних фраз і рекомендувати їм використовувати надійні практики збереження та захисту своїх даних.

1.3.3 Дослідницькі проєкти

В дослідницьких проєктах мнемонічні фрази використовуються для збереження та обробки важливих даних або конфіденційної інформації. Вони можуть забезпечувати безпеку та легкий доступ до цих даних для дослідників та учасників проєктів.

Однак, існують проблеми, пов'язані з можливістю втрати або несанкціонованого доступу до мнемонічних фраз. У разі втрати фрази або доступу до неї можуть виникнути проблеми з відновленням доступу до важливих даних або навіть втратою даних, що мають значення для дослідницького проєкту.

Для подолання цих викликів у дослідницьких проєктах можна використовувати резервні копії мнемонічних фраз, розподілене зберігання фрази серед декількох учасників проєкту або застосування криптографічних методів шифрування для захисту даних. Крім того, можуть розглядатися інноваційні підходи до використання мультіплікативних секретних ділінгів або мультіфакторної аутентифікації для забезпечення безпеки та доступу до мнемонічних фраз у дослідницьких проєктах.

Загалом, використання мнемонічних фраз у різних сферах має свої переваги та недоліки. Застосування належних заходів безпеки, розумних практик збереження та використання мнемонічних фраз, а також інноваційних підходів до аутентифікації може покращити безпеку та зручність використання мнемонічних фраз у практичних застосуваннях.

1.4 Огляд безпекових аспектів роботи з мнемонічними фразами

Огляд безпекових аспектів допоможе розуміти важливість захисту мнемонічних фраз і визначити найкращі практики для забезпечення їх безпеки.

Нижче наведено основні аспекти безпеки, які варто враховувати при роботі з мнемонічними фразами [5].

Мнемонічні фрази містять конфіденційну інформацію, тому необхідно забезпечити їх безпечне зберігання. Втрата фрази або незаконний доступ до неї може призвести до втрати доступу до важливих ресурсів. Важливо обрати надійне фізичне або електронне сховище для збереження мнемонічних фраз і дотримуватися надійних практик забезпечення конфіденційності.

Окрім захисту мнемонічних фраз від несанкціонованого доступу в електронному вигляді, необхідно забезпечити фізичну безпеку пристроїв, на яких зберігаються фрази. Використовуйте надійні пристрої з ефективними методами шифрування та забезпечте фізичну безпеку доступу до цих пристроїв.

Існує ризик втрати мнемонічних фраз або забування їх. Для забезпечення безпеки і можливості відновлення доступу до гаманців чи облікових записів важливо розробити план резервного копіювання та використовувати надійні методи відновлення, такі як багатофакторна аутентифікація.

Атаки з використанням соціальної інженерії є поширеними методами отримання конфіденційної інформації. Зловмисники можуть спробувати переконати користувачів розкрити свої мнемонічні фрази або використовувати інші методи маніпуляції для отримання доступу до гаманців чи облікових записів. Користувачам важливо бути пильними та не розголошувати свої мнемонічні фрази іншим особам.

Шифрування є ефективним методом захисту мнемонічних фраз від несанкціонованого доступу. Використовуйте надійні алгоритми шифрування та захищені протоколи комунікації для захисту мнемонічних фраз у транзиті та зберіганні [6].

Щоб подолати ці виклики та забезпечити високий рівень безпеки при роботі з мнемонічними фразами, можна розглянути наступні шляхи.

Едукація та свідомість. Забезпечення належної освіти користувачів щодо безпеки мнемонічних фраз, виявлення фішингових атак та інших методів атак може допомогти уникнути небезпек і підвищити рівень свідомості щодо захисту.

Використання апаратного забезпечення. Використання апаратних пристроїв, таких як апаратні гаманці, що забезпечують фізичну і криптографічну

безпеку, може додатково підвищити рівень безпеки при роботі з мнемонічними фразами.

Мультіфакторна аутентифікація. Використання мультіфакторної аутентифікації, такої як додаткові підтвердження через SMS, фізичні апаратні токени або біометричні дані, може зробити процес авторизації більш безпечним та складним для несанкціонованого доступу.

Регулярні оновлення та аудит безпеки. Регулярні перевірки та аудит безпеки систем, що використовують мнемонічні фрази, допомагають виявляти потенційні вразливості та приймати заходи для їх виправлення.

Використання блокчейн технологій. В деяких випадках використання блокчейн технологій може забезпечити додатковий рівень безпеки, прозорості та недоступності маніпуляцій при роботі з мнемонічними фразами.

Загалом, забезпечення безпеки мнемонічних фраз вимагає комплексного підходу, що включає технічні, організаційні та поведінкові заходи безпеки. Розуміння потенційних загроз та використання найкращих практик допоможе зберегти конфіденційність та доступ до ресурсів, що використовують мнемонічні фрази.

1.5 Висновки до розділу 1

В даному розділі розглянуті загальні поняття про використання мнемонічних фраз у різних сферах, таких як криптовалютні платформи, фінансові установи та дослідницькі проєкти. Показані переваги мнемонічних фраз, які забезпечують зручне збереження та відновлення важливих ресурсів, але також мають проблеми безпеки, пов'язані з можливістю втрати або несанкціонованого доступу до мнемонічних фраз.

2 ТЕХНІЧНЕ СЕРЕДОВИЩЕ ТА ІНСТРУМЕНТИ РОЗРОБКИ

2.1 Огляд мови програмування Python

Python є однією з найпопулярніших і широко використовуваних мов програмування у світі. Вона була створена у 1991 році Гвідо ван Россумом, і з тих пір набула значної популярності в розробці програмного забезпечення, наукових дослідженнях, веб-розробці та багатьох інших областях.

Однією з найважливіших особливостей Python є його простота і легкість вивчення. Мова має зрозумілий та лаконічний синтаксис, що дозволяє програмістам швидко вирішувати задачі та писати чистий, зрозумілий код. Python підтримує декілька парадигм програмування, включаючи процедурний, об'єктно-орієнтований та функціональний підходи, що робить його гнучким і пристосованим до різних стилів програмування [7].

Один з найбільших переваг Python – це велика кількість доступних бібліотек та модулів. Це дозволяє розробникам ефективно виконувати різноманітні завдання без необхідності писати код з нуля. Наприклад, бібліотека NumPy надає потужні функції для роботи з числовими масивами, бібліотека Pandas – для обробки та аналізу даних, а бібліотека Matplotlib – для візуалізації даних. Крім того, Python має вбудовану бібліотеку стандартних модулів, які покривають широкий спектр функціональних можливостей, таких як робота з файлами, мережеве програмування, обробка регулярних виразів та багато іншого [8].

Програми, написані на Python, можуть запускатись на різних операційних системах, без необхідності вносити значні зміни в код. Популярність Python значно збільшується завдяки його використанню у сферах штучного інтелекту, машинного навчання, обробки природної мови та інших сферах, де потрібно розв'язувати складні завдання. Багато компаній та організацій обирають Python

для розробки своїх продуктів та проєктів, що створює широкі можливості для роботи і розвитку в цій галузі.

У підсумку, Python є потужним і простим у використанні інструментом програмування, що надає широкі можливості для розробки різноманітних проєктів. Велика кількість бібліотек і активна спільнота розробників роблять Python привабливим вибором для багатьох програмістів. Навчання Python відкриває двері до великої кількості можливостей і дозволяє реалізувати різноманітні проєкти у світі програмування та наукових досліджень.

2.2 Технічне середовище для розробки Python

Python, як одна з найпопулярніших мов програмування, має багато різноманітних інструментів та середовищ, які полегшують процес розробки і дозволяють програмістам працювати більш продуктивно і ефективно. У даному розділі розглянуто основні компоненти технічного середовища для розробки Python, включаючи редактори коду, інтегровані середовища розробки (IDE), системи керування пакетами, віртуальні середовища та інші корисні інструменти.

2.2.1 Редактори коду

Редактори коду є базовими інструментами для написання і редагування коду Python. Вони надають основні функції форматування, підсвічування синтаксису, автодоповнення та відлагодження коду. Деякі популярні редактори коду для Python: Visual Studio Code, Sublime Text, Atom, PyCharm Community Edition та інші. Вибір редактора коду залежить від особистих вподобань та потреб розробника.

2.2.2 Інтегровані середовища розробки (IDE)

Інтегровані середовища розробки – це розширені інструменти, які надають розробникам більш широкий функціонал для роботи з кодом Python. Вони мають додаткові можливості, такі як налагодження, керування проєктами, підтримка систем контролю версій та інші. Два популярні варіанти IDE для розробки Python – це PyCharm Professional Edition і Spyder. Ці середовища надають розширені функції та інструменти, що роблять процес розробки більш зручним та ефективним.

2.2.3 Системи керування пакетами

Системи керування пакетами дозволяють легко встановлювати, оновлювати та керувати залежностями пакетів Python. Найпопулярнішою системою керування пакетами є pip – стандартний пакетний менеджер для Python. За допомогою pip можна швидко встановлювати пакети з Python Package Index (PyPI) та контролювати версії пакетів у проєкті. Інші популярні системи керування пакетами включають Anaconda і Poetry, які також надають додаткові функції, такі як управління віртуальними середовищами та проєктами.

2.2.4 Віртуальні середовища

Віртуальні середовища дозволяють створювати ізольовані простори для розробки, де можна встановлювати окремі версії пакетів Python та уникнути конфліктів між ними. Вони корисні при розробці декількох проєктів з різними залежностями. Дві поширені бібліотеки для роботи з віртуальними середовищами – це virtualenv і conda. Вони дозволяють створювати, активувати та керувати віртуальними середовищами для ізоляції проєктів та їх залежностей.

2.2.5 Інші корисні інструменти

Крім основних компонентів, існують інші корисні інструменти, що полегшують процес розробки Python. Наприклад, системи контролю версій, такі як Git, дозволяють керувати версіями коду та спільно працювати з командою розробників. Також можна використовувати бібліотеки та інструменти, такі як pytest для тестування, pylint для аналізу коду, Jupyter Notebook для інтерактивного програмування та документування, і багато інших.

В цьому розділі розглянуто основні компоненти технічного середовища для розробки Python, включаючи редактори коду, інтегровані середовища розробки, системи керування пакетами, віртуальні середовища та інші корисні інструменти. Правильний вибір цих компонентів залежить від потреб розробника та специфіки проєкту.

2.3 Приклади технічних інструментів для розробки

2.3.1. PyCharm

PyCharm, розроблений компанією JetBrains, є потужним інтегрованим середовищем розробки (IDE) для Python. З його інтуїтивно зрозумілим і добре організованим інтерфейсом, воно становить відмінний вибір як для професіоналів, так і для початківців. PyCharm пропонує багато корисних функцій, включаючи автоматичне завершення коду, відображення синтаксичних помилок, підказки щодо виправлення, підтримку рефакторингу та вбудовану систему відлагодження [9].

Окрім цього, PyCharm інтегрується з популярними системами керування версіями, такими як Git, Mercurial і Subversion, і забезпечує зручний контроль версій прямо з інтерфейсу. Воно також підтримує розширення за допомогою

плагінів, що дозволяє розширити функціональність IDE залежно від потреб користувача [10].

Загалом, PyCharm – це потужний і надійний інструмент для розробки на Python. Він допомагає розробникам написати, відлагодити та керувати версіями свого коду з високою продуктивністю і ефективністю. Незалежно від вашого рівня досвіду, PyCharm стане відмінним помічником у ваших проєктах розробки на Python.

2.3.2. Visual Studio Code

Visual Studio Code (VS Code) – популярне інтегроване середовище розробки для Python. Воно поєднує легкість використання з потужними можливостями, задовольняючи потреби розробників будь-якого рівня досвіду. Завдяки широкому вибору розширень і плагінів, доступних у маркетплейсі, VS Code можна налаштувати під свої потреби, додавши функціональність, як підсвічування синтаксису, автоматичне завершення коду, лінтери та інструменти аналізу коду.

VS Code має вбудовану підтримку Git, що полегшує спільну роботу над проєктами. Ви можете здійснювати коміти, переглядати гілки та виконувати злиття безпосередньо з інтерфейсу VS Code. Автоматичне вирішення конфліктів злиття допомагає плавно розвивати проєкт. VS Code також має вбудовану консоль, що дозволяє виконувати Python-скрипти прямо в середовищі розробки. Це забезпечує швидкість та ефективність, дозволяючи перевіряти та тестувати код без виходу з VS Code.

В цілому, Visual Studio Code – зручний, розширюваний та потужний інструмент для розробки в Python. Він надає зручне середовище для написання, відлагодження та керування версіями коду, відповідаючи потребам розробників з будь-яким рівнем досвіду.

2.3.3. Jupyter Notebook

Jupyter Notebook є інтерактивним середовищем, яке дозволяє вам створювати та виконувати код у вигляді блокнотів. Також можна поєднувати код, текстові описи та візуалізації в одному документі, що робить його ідеальним інструментом для дослідження, документування та демонстрації коду. Jupyter Notebook набув популярності серед дослідників даних, аналітиків та викладачів, оскільки він дозволяє зберігати не тільки код, але й контекст його виконання.

2.3.4 Sublime Text

Sublime Text є швидким та легким редактором коду, який підтримує Python та багато інших мов програмування. Він має потужні функції, такі як підсвічування синтаксису, швидкий пошук та заміна, розширення та інші корисні інструменти. Sublime Text також підтримує використання плагінів, що дозволяє налаштувати його для конкретного проєкту.

2.3.5. Віртуальні середовища

Віртуальні середовища, такі як virtualenv, conda та pyenv, дозволяють створювати ізольовані середовища для Python-проєкту. Вони дають можливість управляти залежностями та версіями пакетів, що спрощує розробку і розгортання проєкту. Завдяки цьому можна легко створювати, активувати та використовувати віртуальні середовища для кожного проєкту окремо.

2.3.6. Системи контролю версій

Git, Mercurial та SVN є популярними системами контролю версій, які дозволяють вам керувати різними версіями вашого коду. Вони забезпечують можливість стежити за змінами, робити злиття гілок та відновлення попередніх версій. Системи контролю версій є важливим інструментом для спільної роботи над проєктами та забезпечення безпеки та цілісності кодової бази.

2.3.7. Qt Designer

Qt Designer є графічним інструментом для створення інтерфейсів користувача (UI) на основі фреймворку Qt. Він надає зручний інтерфейс для створення, редагування та налагодження UI-елементів, таких як вікна, кнопки, поля введення і багато інших. Qt Designer дозволяє візуально створювати UI, що значно спрощує розробку програм з графічним інтерфейсом в Python. Після створення UI в Qt Designer, можна експортувати його в код Python і використовувати у своєму проєкті [12].

2.3.8. Qt Linguist

Qt Linguist є інструментом для локалізації та перекладу програм, розроблених з використанням фреймворку Qt. Він надає можливості для перекладу текстових ресурсів, таких як повідомлення, мітки, кнопки, інтерфейси користувача і багато інших. За допомогою Qt Linguist можна легко створювати та керувати перекладами програмного забезпечення, що дозволяє йому бути доступним для користувачів з різних країн та мовних середовищ.

2.4 Загальний огляд бібліотек для розробки

У сучасному світі програмування, використання бібліотек є невід'ємною частиною розробки програмного забезпечення. Бібліотеки - це набори попередньо написаних кодів, які розробники можуть використовувати для покращення продуктивності та розширення функціональності своїх програм. Вони надають готові рішення для різних завдань і проблем, що дозволяє розробникам економити час і зусилля на реалізацію власних функцій.

Важливість використання бібліотек полягає в тому, що вони дозволяють розробникам уникнути повторного написання коду для загальних завдань. Замість цього, вони можуть просто підключити відповідну бібліотеку і використовувати її функціонал. Це сприяє прискоренню процесу розробки, підвищенню продуктивності та поліпшенню якості програмного забезпечення.

На сьогоднішній день існує безліч популярних бібліотек, які підтримуються активною спільнотою розробників і використовуються у різних сферах програмування. Деякі з них відносяться до загального призначення, тоді як інші спеціалізуються на конкретних областях.

2.4.1 PyQt

PyQt є однією з найпопулярніших бібліотек для розробки графічного інтерфейсу користувача (GUI) на мові програмування Python. Вона є прив'язкою Python до фреймворку Qt, який надає потужні інструменти для створення професійних додатків з графічним інтерфейсом [11].

Одним з головних переваг PyQt є його багатий функціонал і можливості. Вона надає доступ до повного набору інструментів Qt, таких як віджети, графічні елементи, стилізація, події та багато іншого. Завдяки цьому, розробники можуть створювати інтерактивні інтерфейси, які привертають увагу користувачів і забезпечують зручну навігацію.

PyQt підтримує різні типи віджетів, включаючи кнопки, поля введення, списки, таблиці, меню і багато інших. Крім того, вона надає можливість створювати власні віджети та керувати їх виглядом і поведінкою. За допомогою PyQt розробники можуть реалізувати багатофункціональні інтерфейси з анімацією, переходами і спеціальними ефектами.

Крім того, PyQt підтримує кросплатформенність, що означає, що програми, розроблені з використанням PyQt, можуть працювати на різних операційних системах, таких як Windows, macOS і Linux. Це дозволяє розробникам створювати універсальні додатки, які можуть бути запущені на різних платформах без необхідності великих змін.

Загалом, PyQt є потужним інструментом для розробки графічних інтерфейсів у Python. Вона надає розробникам широкі можливості для створення професійних інтерфейсів з використанням багатофункціональних віджетів та графічних елементів. Її популярність і підтримка спільноти роблять PyQt привабливим вибором для розробки різноманітних додатків, починаючи від простих інструментів до складних програм з графічним інтерфейсом.

2.4.2 Mnemonic

Ще однією цікавою бібліотекою є mnemonic. Вона призначена для роботи з мнемонічними фразами, які використовуються для створення та відновлення криптографічних ключів. Mnemonic надає зручні функції для генерації мнемонічних фраз, перетворення їх у відповідні криптографічні ключі і здійснення різних операцій з ними. Це значно спрощує роботу з криптографічними ключами та забезпечує більш безпечний спосіб їх зберігання та використання.

2.4.3 Web3

Окрему увагу варто звернути на бібліотеку web3. Вона призначена для взаємодії з блокчейнами та контрактами. Web3 надає зручний інтерфейс для з'єднання з різними блокчейнами, виконання транзакцій, взаємодії з контрактами і отримання даних з блокчейн мережі. Вона дозволяє розробникам створювати розподілені додатки (DApps) на базі блокчейн технологій з використанням мови програмування Python [13].

2.5 Технічне завдання проєкту

2.5.1 Опис функціональних та нефункціональних вимог проєкту

Мій проєкт має на меті автоматизувати роботу з криптогаманцями, зосереджуючись на масовій генерації гаманців та перевірці їх балансу. Для досягнення цієї мети, визначено наступні функціональні та нефункціональні вимоги.

Функціональні вимоги:

– масова генерація гаманців. Система повинна забезпечувати можливість генерації великої кількості криптогаманців за короткий період часу. Користувач повинен мати можливість вказати параметри генерації, такі як кількість гаманців та тип криптовалюти;
– перевірка балансу гаманців. Система повинна здійснювати перевірку балансу згенерованих гаманців шляхом зв'язку з блокчейн-мережею. Результат перевірки повинен бути відображений користувачу;
– зберігання результатів. Система повинна зберігати результати генерації гаманців та їх балансів для подальшого використання або аналізу.

Нефункціональні вимоги:

– швидкодія. Система повинна працювати ефективно та забезпечувати

швидку обробку генерації гаманців та перевірки їх балансу;

– надійність. Система повинна бути стабільною і надійною, забезпечуючи точність та цілісність результатів генерації та перевірки.

2.5.2 Вимоги до інтерфейсу та взаємодії з іншими системами

Інтерфейс користувача повинен бути зручним, інтуїтивно зрозумілим та легким у використанні. Користувач повинен мати можливість встановити параметри генерації, запустити процес генерації гаманців та переглянути результати.

Система повинна взаємодіяти з блокчейн-мережею, використовуючи бібліотеку web3. Це дозволить забезпечити взаємодію з різними блокчейнами та контрактами для перевірки балансу гаманців.

2.5.3 Опис архітектури проєкту та його компонентів

Архітектура проєкту заснована на клієнтській моделі. Основні компоненти проєкту наведено нижче.

Користувацький інтерфейс:

– користувач повинен мати можливість встановлювати параметри генерації гаманців та запускати процес генерації;

– можливість перевірки балансу гаманців;

– інтерфейс повинен забезпечувати відображення результатів генерації та стану балансу.

Логіка генерації гаманців:

– цей компонент повинен відповідати за логіку генерації гаманців з заданими параметрами;

використовування бібліотеки mnemonic для генерації мнемонічних

фраз та бібліотеку web3 для взаємодії з блокчейн-мережею.

Логіка перевірки балансу:

– цей компонент повинен відповідати за перевірку балансу гаманців за допомогою взаємодії з блокчейн-мережею;
– використовування бібліотеки web3 для отримання актуальної інформації про баланс гаманців.

Архітектура проєкту повинна бути добре структурованою та модульною, що сприятиме підтримці та розширенню функціональності програми.

2.6 Висновки до розділу 2

У цьому розділі проведено огляд технічного середовища та інструментів для розробки програм на мові програмування Python. Розглянуто загальний огляд мови Python та різні інструменти, які використовуються для розробки, а також редактори коду. Крім того, зроблено огляд трьох важливих бібліотек для розробки: PyQt, яка надає можливості створення графічного інтерфейсу користувача, та бібліотеки Mnemonic і Web3, які дозволяють працювати з мнемонічними фразами та взаємодіяти з блокчейнами та контрактами. Відповідно до технічного завдання проєкту визначено функціональні та нефункціональні вимоги, вимоги до інтерфейсу та взаємодії з іншими системами та описана архітектура проєкту та його компонентів.

3 РОЗРОБКА КЛІЄНТСЬКОЇ ПРОГРАМИ

Реалізація програми є ключовим етапом у процесі розробки. На цьому етапі відображено перехід від проєктування до практичної реалізації моєї програми. У даному розділі я розглянув основні кроки та процеси, пов'язані з реалізацією програми для автоматизації роботи з криптогаманцями.

3.1 Встановлення необхідних компонентів

3.1.1 Середовище розробки

Першим кроком у реалізації програми є встановлення необхідного середовища розробки. Середовище розробки включає інструменти, бібліотеки та засоби, необхідні для написання й виконання програмного коду. Для своєї роботи я обрав PyCharm (див. рис. 3.1).

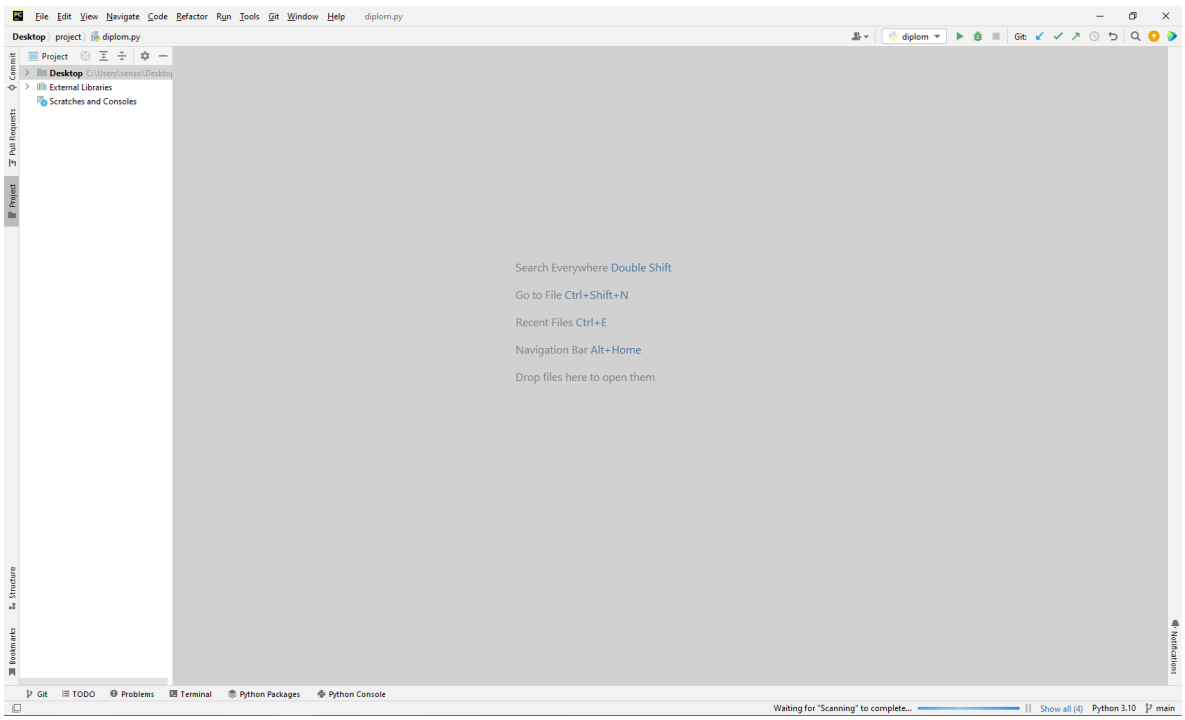

Рисунок 3.1 – Середовище розробки PyCharm

3.1.2 QT designer та QT Linguist

Qt Designer необхідний для проєктування та створення графічних інтерфейсів користувача (див. рис. 3.2).

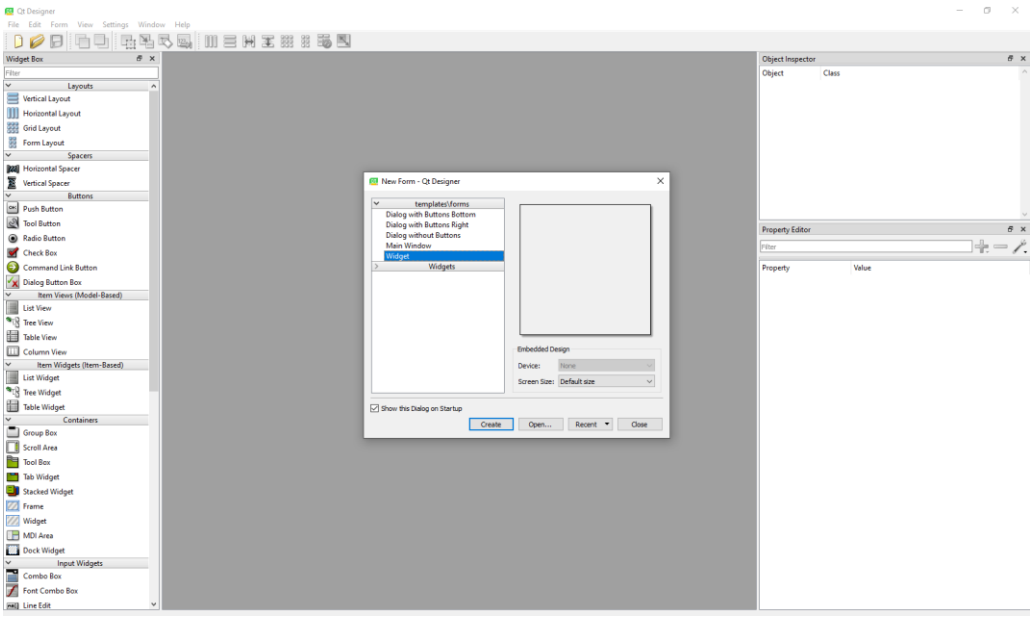

Рисунок 3.2 – Головне меню Qt Designer

Qt Linguist – це інструмент для додавання перекладів (див. рис. 3.3).

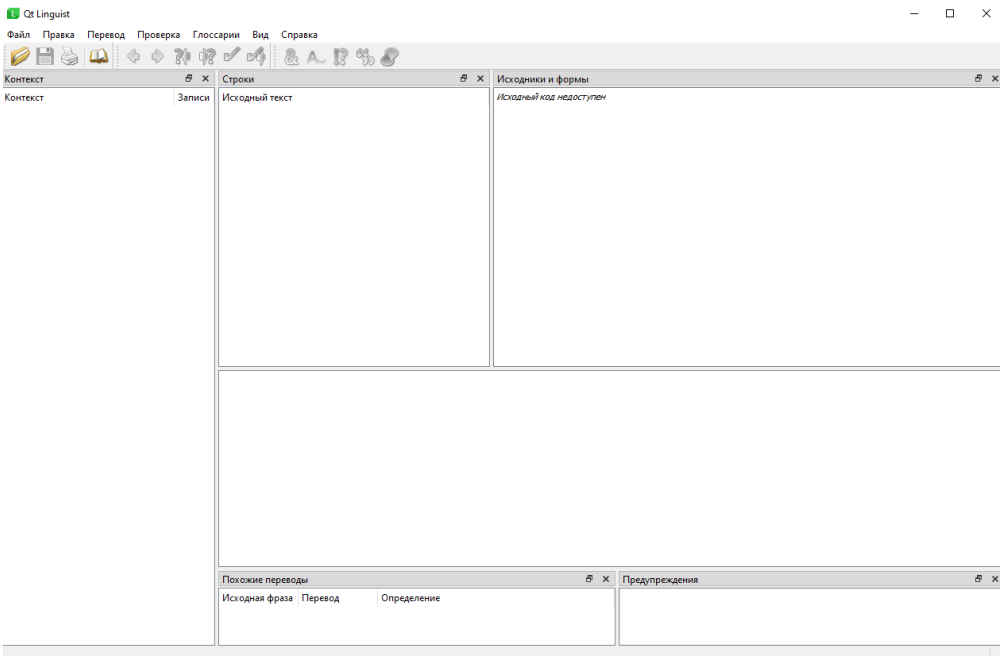

Рисунок 3.3 – Головне меню Qt Linguist

3.1.3 Бібліотеки

У процесі реалізації програми використано різні бібліотеки та інструменти для покращення функціональності та ефективності. А саме:

- для роботи з графічним інтерфейсом встановлюємо бібліотеку PyQt6, яка надає інструменти для розробки віконних додатків. Встановлюємо за допомогою команди: pip install PyQt6;
- для роботи з мнемонічними фразами використовуємо бібліотеку mnemonic. Встановлюємо за допомогою команди: pip install mnemonic;
- для взаємодії з блокчейнами та контрактами – бібліотеку web3. Встановлюємо за допомогою команди: pip install web3;
- для відправки запитів було використано бібліотеку Requests. Встановлюємо за допомогою команди: pip install requests.

3.2 Створення вікон програми

Створення вікон програми є важливим етапом в розробці програмного забезпечення. У цьому розділі розглянуто процес створення різних вікон програми, таких як головне меню, меню налаштувань, меню перевірки балансу та меню генерації.

3.2.1 Головне меню

Головне меню є ключовим елементом багатьох програм, яке містить основні функції та опції доступні користувачу. В моїй програмі, на головній сторінці 3 кнопки, які відповідатимуть за перехід до однієї з наступних сторінок, а саме, «Генерація», «Перевірка балансу», «Налаштування» (див. рис. 3.4).

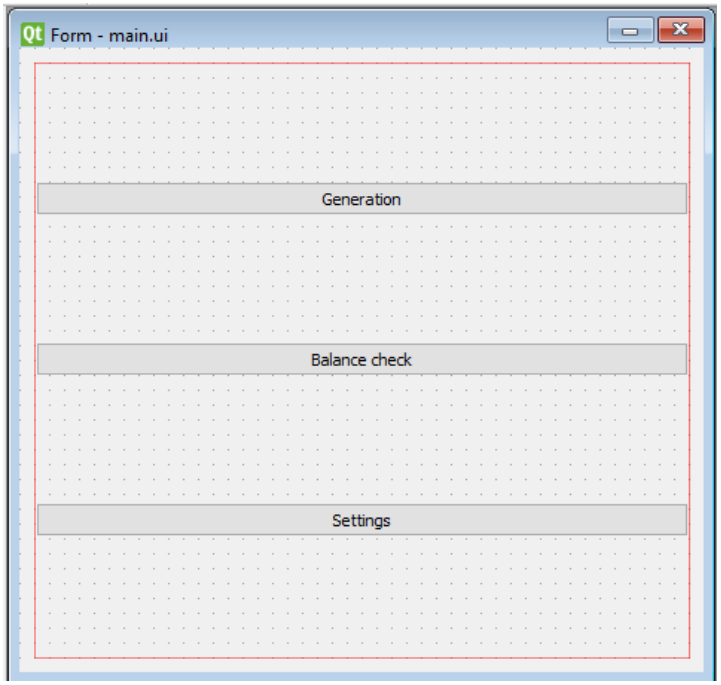

Рисунок 3.4 – Головне меню

3.2.2 Меню налаштувань

Меню налаштувань дозволяє користувачам налаштувати параметри та опції програми за їх власними потребами. В моїй програми буде можливість налаштування зовнішнього вигляду та мови програми (див. рис. 3.5).

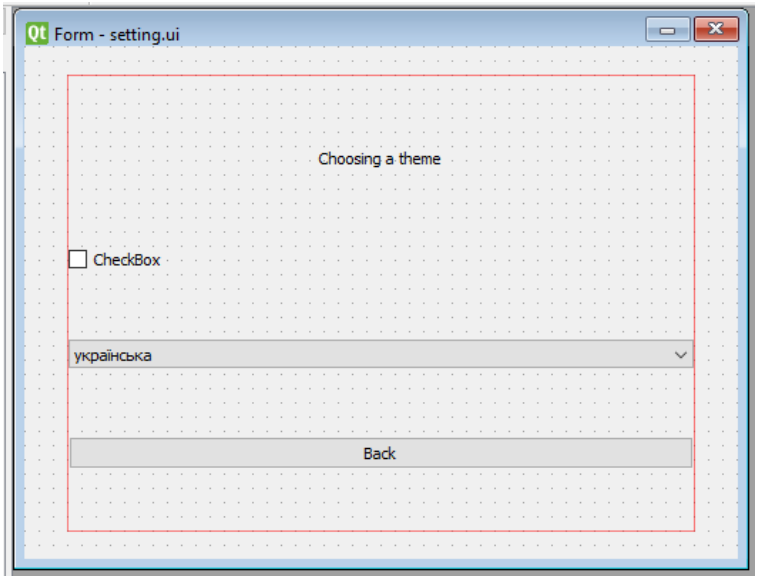

Рисунок 3.5 – Меню налаштувань

3.2.3 Меню вибору категорії перевірки балансу

Це меню відповідає за вибір критеріїв за якими буде здійснюватися перевірка балансу. Створюємо пункти меню з вибором, а саме: перевірка по публічному адресу, по приватному ключу, по мнемонічній фразі. Після натискання на одну з кнопок, буде відкрито наступне меню перевірки балансу (див. рис. 3.6).

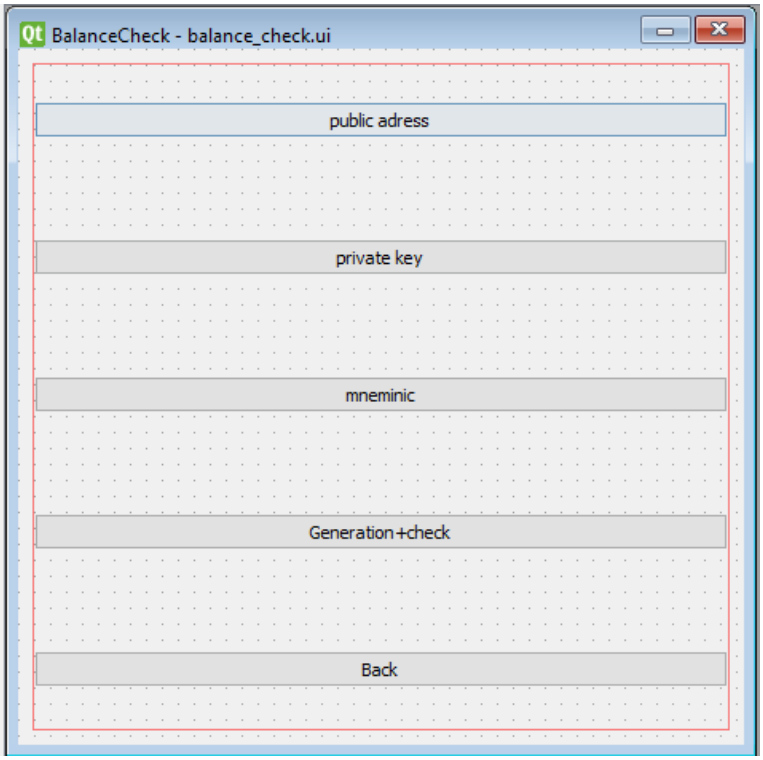

Рисунок 3.6 – Меню вибору категорій перевірки балансу

3.2.4 Меню перевірки балансу

Меню перевірки балансу дозволяє користувачам перевіряти баланс гаманців. Додано поле для введення тексту та відображення результату перевірки. Також додано три кнопки: для зчитування даних із файлу, для збереження результатів у файл та для запуску перевірки. Для зручності додано індикатор виконання (див. рис. 3.7).

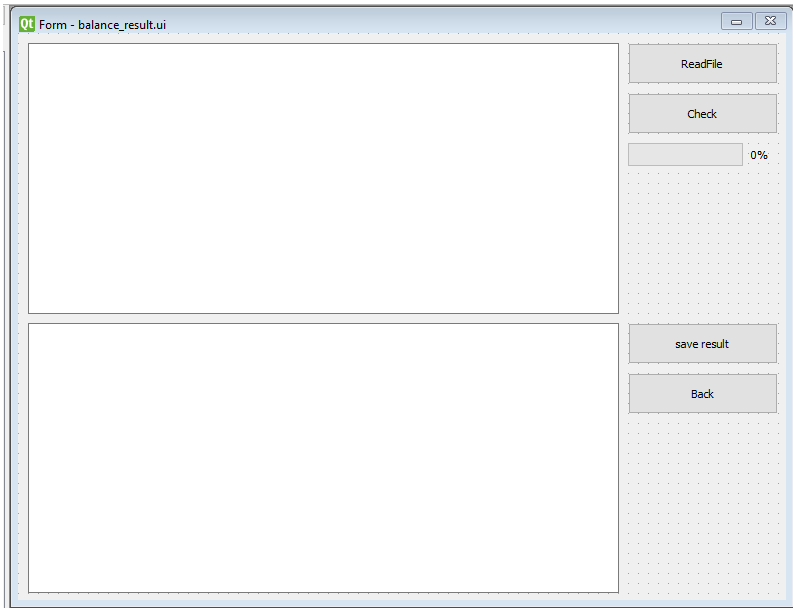

Рисунок 3.7 – Меню перевірки балансу

3.2.5. Меню вибору типу генерації

Меню вибору типу генерації дозволяє користувачам обрати формат генерації криптогаманців. Додано поле для введення кількості гаманців, яке потрібно згенерувати, а також формат генерованих даних, наприклад лише приватні ключі, тільки мнемонічні фрази або рядки, що містять повні дані про гаманець, у тому числі публічний ключ (див. рис. 3.8). Після натискання кнопки запуску, повинно буде відкриватись меню з результатом генерації.

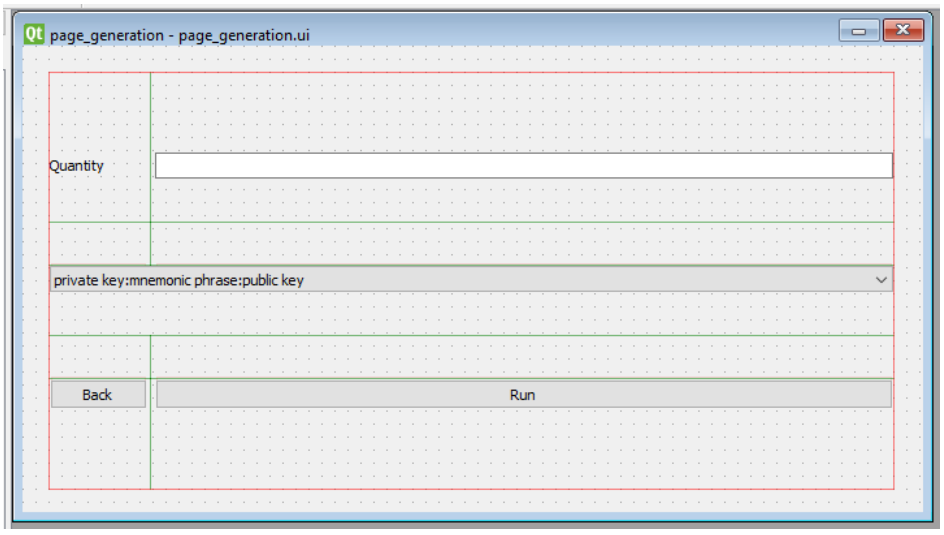

Рисунок 3.8 – Меню вибору типу генерації

3.2.6. Меню результатів генерації

Це меню відображатиме результати генерації. Вікно містить текстове поле, в якому відображатиметься результат виконання генерації та кнопку збереження даних з текстового поля у файл (див. рис. 3.9).

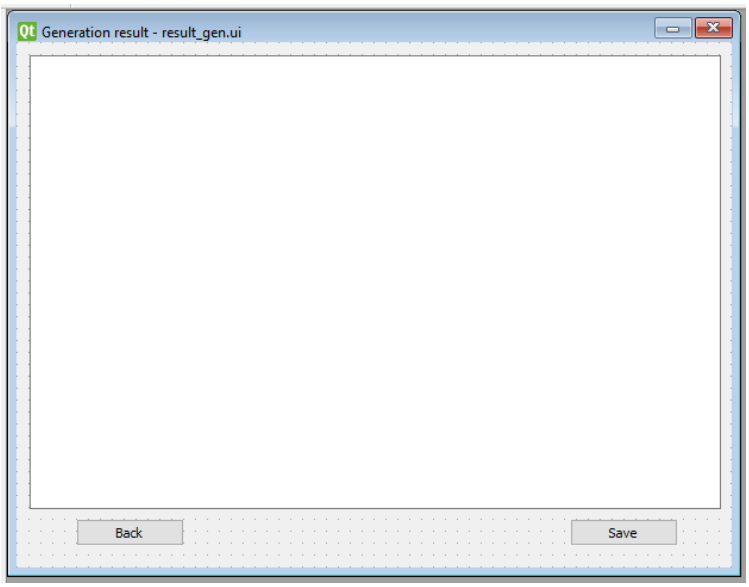

Рисунок 3.9 – Меню результатів генерації

3.3 Реалізація функціональності

Реалізація функціональності є важливим етапом у процесі розробки програмного забезпечення. У цьому розділі розглянуто процес реалізації функціональності для моєї програми, яка автоматизує роботу з криптогаманцями.

3.3.1. Розробка модулів

Розпочинаємо з розробки необхідних модулів, класів та функцій, які будуть використовуватися у програмі. Створюю основну структуру моєї

програми та визначаю, які компоненти будуть взаємодіяти між собою.

```python
# Form implementation generated from reading ui file 'main.ui'
#
# Created by: PyQt6 UI code generator 6.5.0
#
# WARNING: Any manual changes made to this file will be lost when pyuic6 is
# run again.  Do not edit this file unless you know what you are doing.

from PyQt6 import QtCore, QtGui, QtWidgets


2 usages
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(460, 420)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Policy.Preferred, QtWidgets.QSizePolicy.Policy.Preferre
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(MainWindow.sizePolicy().hasHeightForWidth())
        MainWindow.setSizePolicy(sizePolicy)
        MainWindow.setMinimumSize(QtCore.QSize(460, 420))
        MainWindow.setMaximumSize(QtCore.QSize(460, 420))
        self.widget = QtWidgets.QWidget(parent=MainWindow)
        self.widget.setGeometry(QtCore.QRect(10, 10, 441, 401))
        self.widget.setObjectName("widget")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.widget)
        self.verticalLayout.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout.setObjectName("verticalLayout")
```

Рисунок 3.10 – Приклад згенерованого модуля інтерфейсу

Спочатку треба конвертувати створені раніше файли з інтерфейсом. Для цього виконуємо команду `pyuic6 main.ui -o main_ui.py`. Ця команда виконує конвертацію файлу `main.ui`, який є файлом інтерфейсу, створеним у програмі Qt Designer для PyQt6, у вихідний код Python. Результат цієї команди буде збережений у файлі з назвою `main_ui.py` (див. рис. 3.10). Повторюємо цю команду для кожного файлу інтерфейсу.

Наступний крок, це робота з Checkbox. `QCheckBox` є класом у бібліотеці PyQt6, який представляє елемент керування "Прапорець" (Checkbox) у графічному інтерфейсі користувача. Він надає можливість користувачам встановлювати або скасовувати прапорець, що вказує на присутність або відсутність певного стану або опції. Зазвичай `QCheckBox` відображається як малюнок з прапорцем, який може бути встановлений (включений) або скасований (вимкнений). Користувач може клацнути на `QCheckBox`, щоб

змінити його стан.

Оскільки мені не подобається зовнішній вигляд "Прапореця", було створено новий клас, який замінить стандартний клас QCheckBox. Новий клас `AnimatedToggle` успадковує функціональність від `QCheckBox` і розширює його, додаючи анімаційний вигляд для перемикача. Він використовує різні властивості, методи та анімації для забезпечення зміни вигляду та відображення стану перемикача у графічному інтерфейсі PyQt6 (див. рис. 3.11).

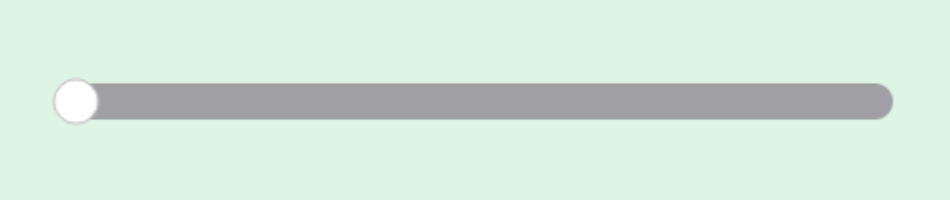

Рисунок 3.11 – Створений анімований перемикач

Останні два файли, що визначають стилі (CSS) для елементів графічного інтерфейсу PyQt6, таких як `QPushButton`, `QMainWindow`, `QLineEdit`, `QLabel` та `QComboBox`, дозволяють налаштувати зовнішній вигляд програми в залежності від обраної теми. Основні властивості стилів включають фоновий колір (`background`), шрифт (`font`), колір тексту (`color`), товщину та стиль рамки (`border-width` та `border-style`), колір рамки (`border-color`), вирівнювання тексту (`text-align`), радіус закруглення кутиків рамки (`border-radius`) та інші. Перший файл представляє світлу тему, в якій використовуються яскраві та легкі кольори. Другий файл для темної теми містить параметри, які створюють затемнений та більш контрастний вигляд для елементів (див. рис. 3.12).

Використання різних файлів стилів дозволяє легко змінювати вигляд програми в залежності від поточної теми, що вибрана користувачем або налаштована програмою за замовчуванням. Це надає можливість створити гнучкий та персоналізований інтерфейс, який відповідає вимогам та вподобанням користувача.

style_light.py

```
STYLE_SHEET_LIGHT = '''
QPushButton {
    background: #8EC3B0;
    font: bold 28px;
    color: #E7F6F2;
    border-width: 2px;
    border-style: solid;
    border-color: #9ED5C5;
    text-align: center;
    outline: none;
}
QPushButton:hover {
    color: #8EC3B0;
    border-color: #E7F6F2;
    background: #BCEAD5;
    outline: none;
}
QMainWindow{
    background-color: #DEF5E5;
}

QLineEdit{
    font-family: Rubik;
    font-size: 28px;
    background: #8EC3B0;
    color: #E7F6F2;
    border-width: 2px;
```

Рисунок 3.12 – Приклад створеного коду, що містить стилі CSS

3.3.2. Кодування функцій

Кодування функцій є ключовим етапом у розробці програми, де перейдемо до написання конкретного коду для реалізації функціональності програми. На цьому етапі використано вже розроблені модулі та функції, які надають необхідні засоби для виконання потрібних завдань.

Використання потоків у PyQt (або будь-якій іншій GUI-бібліотеці) допомагає уникнути зависання або загальмування інтерфейсу користувача (UI) під час виконання довгих або обчислювально важких операцій.

У багатьох GUI-бібліотеках, включаючи PyQt, головний потік виконується

в основному циклі подій (event loop), який відповідає за обробку подій від користувача, таких як натискання кнопок чи переміщення миші. Якщо виконати довгу або обчислювально важку операцію безпосередньо в головному потоці, вона може зайняти значну частину часу, що спричинить затримки в обробці подій і зробить інтерфейс несприйнятливим для користувача.

Використання додаткових потоків дозволяє виконувати ці довгі або обчислювально важкі операції в окремому потоці, не блокуючи головний потік UI. Це дозволяє UI залишатися реактивним і продовжувати обробку подій користувача незалежно від виконання операцій у фоновому потоці.

3.3.2.1 Функція генерації

У моєму коді (див. рис. 3.13) `MyThread` є підкласом `QThread`, який надає можливість виконувати код у відокремленому потоці.

```
class MyThread(QtCore.QThread):
    def __init__(self, idx, amount):
        super().__init__()
        self.idx = idx
        self.amount = amount

    about_new_log = QtCore.pyqtSignal(str)

    def run(self):
        for i in range(self.amount):
            my_mnemonic = Mnemonic("english")
            words = my_mnemonic.generate(128)
            w3 = Web3()
            w3.eth.account.enable_unaudited_hdwallet_features()
            account = w3.eth.account.from_mnemonic(words, account_path="m/44'/60'/0'/0/0")
            if self.idx == 0:
                self.about_new_log.emit(f'{words}:{w3.toHex(account.key)}:{account.address}\n')
            elif self.idx == 1:
                self.about_new_log.emit(f'{w3.toHex(account.key)}\n')

            elif self.idx == 2:
                self.about_new_log.emit(f'{words}\n')
```

Рисунок 3.13 – Функція генерації криптогаманців

Метод `run` у цьому підкласі містить код, який буде виконуватись у фоновому потоці. Таким чином, при запуску цього потоку, код, який знаходиться у `run`, буде виконуватись паралельно з головним потоком, дозволяючи уникнути зависання UI під час виконання довгих або обчислювально важких операцій.

Завдяки використанню сигналу `about_new_log`, мій код може сповіщати головний потік про результати обчислень, що відбуваються у фоновому потоці. Це дає можливість оновлювати інтерфейс користувача з отриманими результатами або відображати проміжні результати в режимі реального часу без блокування головного потоку.

Створено підклас `MyThread`, який успадковує від `QtCore.QThread`. Він має такі методи:

а) `__init__(self, idx, amount)`: це конструктор класу, який приймає два аргументи `idx` і `amount` і зберігає їх як внутрішні змінні `self.idx` і `self.amount`;

б) `about_new_log = QtCore.pyqtSignal(str)`: це сигнал, який відправляє рядок (`str`). Він буде використовуватися для сповіщення про новий журнальний запис;

в) `run(self)`: цей метод виконується під час запуску потоку. У циклі `for` від 0 до `self.amount` виконуються наступні дії:

1) створюється екземпляр `Mnemonic` з параметром "english";

2) генерується 128-бітна послідовність слів (`words`) за допомогою об'єкта `my_mnemonic`;

3) створюється екземпляр `Web3`;

4) включаються неверифіковані функції гаманця за допомогою `w3.eth.account.enable_unaudited_hdwallet_features()`;

5) створюється обліковий запис (`account`) за допомогою `w3.eth.account.from_mnemonic(words, account_path="m/44'/60'/0'/0/0")`;

6) в залежності від значення `self.idx` відправляється сигнал `about_new_log` з відповідним рядком;

7) якщо `self.idx` дорівнює 0, то сигнал містить рядок, який складається з послідовності слів (`words`), шістнадцяткового представлення ключа (`w3.toHex(account.key)`) та адреси облікового запису (`account.address`);

8) якщо `self.idx` дорівнює 1, то сигнал містить шістнадцяткове представлення ключа (`w3.toHex(account.key)`);

9) якщо `self.idx` дорівнює 2, то сигнал містить послідовність слів (`words`).

Отже, після запуску потоку будуть генеруватись дані (слова, ключі та адреси облікових записів) залежно від значення `self.idx`, і ці дані будуть передаватись через сигнал `about_new_log`.

3.3.2.2 Функція перевірки балансу

Створено підклас `MyThread2` (див. рис. 3.14), який також успадковує від `QThread`. Цей підклас містить логіку для виконання фонових операцій у відокремленому потоці. У конструкторі `__init__` класу `MyThread2` приймаються параметри `arr` (масив) і `option` (опція). Ці параметри ініціалізуються відповідними атрибутами класу.

Два сигнали `about_new_log` і `progress_bar` також оголошуються як атрибути класу. `about_new_log` використовується для відправки повідомлень про результати обчислень, а `progress_bar` використовується для оновлення панелі прогресу.

У методі `run` виконується фактична робота у фоновому потоці. Перший крок – ініціалізація об'єкту `Web3` і увімкнення функціональності гаманця без перевірки аудиту.

Далі визначена функція `res`, яка приймає один параметр `adrs`. У цій функції відбувається виконання запиту до різних блокчейн-експлорерів для отримання балансу акаунта за вказаною адресою. Отримані результати виводяться на консоль та емітуються через сигнал `about_new_log`.

```python
def run(self):
    value = 0
    w3 = Web3()
    w3.eth.account.enable_unaudited_hdwallet_features()

    def res(adrs):
        urls = ['etherscan.io', 'bscscan.com', 'polygonscan.com']
        apikeys = ['EDUMG27SUB3JTJJ6VTFTYS22254IK5CTVR', 'BJY8BH9M9GM9Q789BDQ4YGPQMRPP92Y5V8', 'SQ6RSGX72VY824AV8SJHMTISK49QMKT6D4']
        self.about_new_log.emit(f'{adrs}\n')
        for url, apikey in zip(urls, apikeys):
            params = {"module": "account", "action": "balance", "address": str(adrs), "tag": "latest",
                      "apikey": apikey}
            response = requests.get(f'https://api.{url}/api', params=params).json()
            total_balance = int(response["result"]) / (10 ** 18)
            print(f'{url}: ', total_balance)
            self.about_new_log.emit(f'{url}: {total_balance}\n')

    if self.option == 'public adress':
        for address in self.arr:
            res(address)
            value += round(100 / len(self.arr))
            self.progress_bar.emit(value)
        self.progress_bar.emit(100)

    if self.option == 'private key':
        for address in self.arr:
            account = w3.eth.account.privateKeyToAccount(address)
            res(account.address)
            value += int(100 / len(self.arr))
            self.progress_bar.emit(value)
        self.progress_bar.emit(100)

    if self.option == 'mneminic':
        for address in self.arr:
            account = w3.eth.account.from_mnemonic(address, account_path="m/44'/60'/0'/0/0")
            res(account.address)
            value += int(100 / len(self.arr))
            self.progress_bar.emit(value)
```

Рисунок 3.14 – Функція перевірки балансу

У методі `run` подальший код залежить від значення параметра `option`. Якщо `option` дорівнює 'public adress', 'private key' або 'mneminic', відповідно, то виконується відповідна гілка коду. В кожній гілці цикл `for` обходить масив `arr` і викликає функцію `res` для кожного елементу. Після цього оновлюється значення `value`, яке використовується для розрахунку прогресу, і емітуються оновлення прогресу через сигнал `progress_bar`. В кінці кожної гілки коду емітуються сигнали `progress_bar` з значенням 100, що означає завершення операцій.

Отже, цей код створює окремий потік виконання, який виконує запити до блокчейн-експлорерів для отримання інформації про баланси акаунтів за вказаними адресами, а результати передаються в основний потік за допомогою сигналів.

3.3.2.3 Функція зміни мови

Функція `change_language` використовується для зміни мови програми. Вона приймає параметр `language_code`, який вказує на обрану мову. У функції створюється об'єкт `translator` класу `QTranslator`. Далі виконується перевірка значення `language_code`. Якщо `language_code` дорівнює 'Ukrainian', то встановлюється переклад на українську мову. Це виконується за допомогою методів `installTranslator` та `load` об'єкта `app` (екземпляр `QApplication`), які встановлюють перекладовий файл `myapp_uk.qm` і встановлюють українську локалізацію. Якщо `language_code` дорівнює 'English', то здійснюється зміна на англійську мову. У цьому випадку, за допомогою методу `removeTranslator`, перекладовий об'єкт `translator` видаляється з `app` і встановлюється англійська локалізація.

Після зміни мови, значення `language_code` зберігається в об'єкті `setting` (це об'єкт налаштувань), використовуючи метод `setValue` (див. рис. 3.15).

```
def change_language(self, language_code):
    translator = QTranslator(self)

    if language_code == 'Ukrainian':
        app.installTranslator(translator)
        translator.load("translations/myapp_uk.qm")
        locale = QLocale('uk')
        QLocale.setDefault(locale)

    if language_code == 'English':
        app.removeTranslator(translator)
        locale = QLocale('en')
        QLocale.setDefault(locale)
    print(language_code)
    self.setting.setValue("language", language_code)
```

Рисунок 3.15 – Функція зміни мови

3.3.2.4 Функція зміни кольорової теми

Функція `color_change` використовується для зміни стилів вікна програми. Вона отримує параметр `selected`, який вказує, чи було вибрано новий стиль. Якщо `selected` має значення `True`, то встановлюється темний стиль програми. Це досягається за допомогою методу `setStyleSheet` об'єкта `app`, якому передається змінна `STYLE_SHEET`. Крім того, значення `window style` в об'єкті `setting` зберігається як `STYLE_SHEET`. Якщо `selected` має значення `False`, то встановлюється світлий стиль, використовуючи змінну `STYLE_SHEET_LIGHT`. Значення `window style` також зберігається в об'єкті setting (див. рис. 3.16). Цей код виконує зміни мови та стилю програми на основі обраних налаштувань, а також зберігає ці налаштування для подальшого використання.

```
1 usage
def color_change(self, selected):
    if selected:
        app.setStyleSheet(STYLE_SHEET)
        self.setting.setValue('window style', STYLE_SHEET)

    else:
        app.setStyleSheet(STYLE_SHEET_LIGHT)
        self.setting.setValue('window style', STYLE_SHEET_LIGHT)

    self.setting.setValue('win status', selected)
```

Рисунок 3.16 – Функція зміни кольорової теми

3.4 Висновки до розділу 3

Отже, у цьому розділі я показав процес розробки клієнтської програми для автоматизації роботи з мнемонічними фразами. Було розглянуто встановлення середовища розробки, розробку функціональності та інтеграцію з використовуваними бібліотеками та інструментами. Крім того, детально

розглянуто процес створення вікон програми. Результатом розділу є створена клієнтська програма, яка забезпечує автоматизацію роботи з криптогаманцями, включаючи генерацію гаманців та перевірку балансу. Розробка такої програми може спростити процес роботи з криптогаманцями та покращити продуктивність користувачів.

ВИСНОВКИ

У цій роботі була проведена детальна розробка та реалізація програми для роботи з мнемонічними фразами криптогаманцями. Починаючи з аналізу предметної області, висвітлення основних понять та використання мнемонічних фраз, розгляду технічного середовища та інструментів розробки, аж до створення вікон програми та реалізації функціональності. Кожен етап був детально проаналізований і описаний.

В процесі дослідження було встановлено, що використання мнемонічних фраз дозволяє забезпечити безпеку і зручність в роботі з криптогаманцями. Також було виявлено, що використання бібліотек PyQt, mnemonic та web3 дозволяє розширити функціональні можливості програми та взаємодіяти з блокчейнами та контрактами.

Результатом роботи є створена програма, яка забезпечує автоматизовану роботу з криптогаманцями, включаючи генерацію гаманців та перевірку балансу. Вона пропонує зручний інтерфейс користувача, розроблений з використанням Python та багатьох корисних інструментів, що значно полегшують процес роботи з криптогаманцями.

Під час розробки було враховано функціональні та нефункціональні вимоги проєкту, а також забезпечено взаємодію з іншими системами. Архітектура програми була детально розроблена, з урахуванням компонентів і модулів. У підсумку, дана робота пропонує глибокий огляд предметної області, високоякісну розробку програмного продукту та детальний опис кроків для реалізації програми для автоматизації роботи з криптогаманцями. Цей проєкт може бути корисним для тих, хто бажає забезпечити ефективну та безпечну роботу з криптовалютними гаманцями.

ПЕРЕЛІК ПОСИЛАНЬ

1. Lee W. M., Lee W. M. Using the metamask chrome extension. *Beginning Ethereum Smart Contracts Programming: With Examples in Python, Solidity, and JavaScript*, 2019. P. 93-126.
2. Suratkar S., Shirole M., Bhirud S. Cryptocurrency wallet: A review. *2020 4th international conference on computer, communication and signal processing (ICCCSP)*. IEEE, 2020. P. 251-264.
3. Wątorek M., Drożdż S., Kwapień J., Minati L., Oświęcimka, P., & Stanuszek, M. Multiscale characteristics of the emerging global cryptocurrency market. *Physics Reports* 901, 2021. 82 p.
4. Fornaro D. Elliptic curve hierarchical deterministic private key sequences: bitcoin standards and best practices. URL: https://www.politesi.polimi.it/bitstream/10589/140112/1/2018_04_Fornaro.pdf (дата звернення: 16.04.2023).
5. Khan A. G., Zahid A. H., Hussain, M., & Riaz, U. Security of cryptocurrency using hardware wallet and qr code. *2019 International Conference on Innovative Computing (ICIC)*. IEEE, 2019. P. 1-19.
6. Er-Rajy L., et al. Blockchain: Bitcoin wallet cryptography security, challenges and countermeasures. *Journal of Internet Banking and Commerce* 22.3, 2017. 29 p.
7. VanRossum G., Drake F. L. *The python language reference*. Amsterdam, Netherlands: Python Software Foundation, 2010. P. 6-43.
8. Van Rossum G., Drake Jr F. L. The python language reference. *Python Software Foundation: Wilmington, DE, USA*, 2014. P. 17-43
9. Islam Q. N. *Mastering PyCharm*. Packt Publishing Ltd, 2015. P. 22-223.
10. Willman J. M. Creating GUIs with Qt Designer. *Beginning PyQt: A Hands-on Approach to GUI Programming with PyQt6*. Berkeley, CA: Apress, 2022. P. 217-258.

11. Harwani B. M. *Qt5 Python GUI Programming Cookbook: Building responsive and powerful cross-platform applications with PyQt*. Packt Publishing Ltd, 2018. P. 23-426.
12. Molkentin D. *The book of Qt 4: The art of building Qt applications*. No Starch Press, 2007. P. 15-374.
13. Lee W. M. Developing Web3 dapps using Python. *Beginning Ethereum Smart Contracts Programming: With Examples in Python, Solidity, and JavaScript*. Berkeley, CA: Apress, 2023. P. 215-239.