Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп’ютерних наук

**Допущено до захисту**
Завідувач кафедри комп’ютерних наук,
кандидат технічних наук, доцент
_____Ірина МАШКІНА
« ___» _____2025 р.

**КВАЛІФІКАЦІЙНА РОБОТА**
**на здобуття освітнього ступеня «бакалавр»**
**Спеціальність 122 Комп’ютерні науки**
**Освітня програма 122.00.01 Інформатика**

**Тема:** РОЗРОБКА АВТОМАТИЗОВАНОГО ІНСТРУМЕНТА ДЛЯ ІНТЕГРАЦІЇ ДАНИХ З РІЗНИХ ДЖЕРЕЛ З МЕТОЮ АНАЛІЗУ ТА ВІЗУАЛІЗАЦІЇ

**Виконав**
студент групи ІНб-1-21-4.0д
Гусак Станіслав Леонідович
_____
(підпис)
**Науковий керівник**
кандидат педагогічних наук,
доцент кафедри комп'ютерних наук
Глушак О.М.
_____
(підпис)

Київ – 2025

Київський столичний університет імені Бориса Грінченка

Факультет інформаційних технологій та математики

Кафедра комп'ютерних наук

**«Затверджую»**

Завідувач кафедри комп'ютерних наук, кандидат технічних наук, доцент

_____Ірина МАШКІНА

« ___» _____2025р.

**ЗАВДАННЯ НА КВАЛІФІКАЦІНУ РОБОТУ БАКАЛАВРА**

**«Розробка автоматизованого інструмента для інтеграції даних з різних джерел з метою аналізу та візуалізації»**

Виконавець – студент групи ІНб-1-21-4.0д, спеціальності 122 Комп'ютерні наук, освітньої програми 122.00.01 Інформатика

**Гусак Станіслав Леонідович**

*1.* Вихідні дані: *Публікації за напрямом інтеграції даних, побудови ETL-процесів, розробки систем візуалізації та обробки фінансової інформації.*

2. Основні завдання: *Здійснити огляд наукових праць за тематикою інтеграції даних і побудови систем аналізу фінансової інформації. Обґрунтувати мету, об'єкт, предмет та наукові основи дослідження. Розробити завдання, структуру та зміст кваліфікаційної роботи. Розробити архітектуру інструменту інтеграції даних із застосуванням відкритих технологій. Реалізувати модулі збору, обробки, збереження даних та вебінтерфейс для візуалізації результатів. Провести аналіз отриманих результатів, скласти висновки. Підготувати тези доповіді за результатами роботи. Пояснювальна записка:* Обсяг – до 45 стор. формату А4 комп'ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.

3. Графічні матеріали: *презентація.*

*4.* Додатки: *не передбачено.*

*5.* Строк подання роботи на кафедру: *«____»* червня *2025 р.*

Науковий керівник _____ дата

к.п.н., доцент кафедри комп'ютерних наук

_____ Глушак О.М.

Виконавець:

_____Гусак С.Л.

_____ дата

**Календарний план**

| № | Етап виконання роботи | Термін виконання | Примітка |
|---|---|---|---|
| 1 | Формування теми дипломної роботи бакалавра та її затвердження | 07.10.2024 – 14.10.2024 | виконано |
| 2 | Аналіз технологій інтеграції даних та огляд літературних джерел | 25.11.2024 – 08.12.2024 | виконано |
| 3 | Постановка задачі, визначення об’єкта, предмета і мети дослідження | 09.12.2024 – 15.12.2024 | виконано |
| 4 | Вибір архітектури інструмента інтеграції даних | 20.01.2025 – 07.02.2025 | виконано |
| 5 | Розробка ETL-процесів: збір, трансформація та збереження даних | 08.02.2025 – 22.02.2025 | виконано |
| 6 | Реалізація серверного API для доступу до даних | 23.02.2025 – 10.03.2025 | виконано |
| 7 | Створення інтерфейсу користувача для візуалізації даних (Streamlit) | 21.03.2025 – 28.03.2025 | виконано |
| 8 | Інтеграція компонентів інструмента, тестування функціоналу | 30.03.2025 – 07.04.2025 | виконано |
| 9 | Оцінка результатів, підготовка висновків | 09.04.2025 – 19.04.2025 | виконано |
| 10 | Передзахист кваліфікаційної роботи | 20.05.2025 | виконано |
| 11 | Внесення правок, затвердження роботи, підготовка матеріалів для захисту | 21.05.2025 – 14.06.2025 | виконано |
| 12 | Захист кваліфікаційної роботи | 19.06.2025 | виконано |

**Здобувач**

студент ІНБ 1-21-4.0д

Гусак Станіслав Леонідович

**Керівник роботи**

к.п.н., доцент кафедри комп'ютерних наук

Глушак О.М

# РЕФЕРАТ бакалаврської роботи

Кваліфікаційна робота: 47 с., 31 рис., 41 посилання.

Актуальність: У сучасних умовах стрімкого зростання обсягів даних та необхідності оперативної обробки інформації питання інтеграції даних із різних джерел є надзвичайно актуальним. Автоматизація процесів збору, обробки та візуалізації даних відіграє ключову роль у побудові аналітичних систем для підтримки прийняття рішень. Реалізація гнучкого інструмента інтеграції фінансових даних із можливістю подальшої аналітики є важливою задачею в умовах цифрової трансформації.

Об'єкт дослідження: Інформаційні потоки фінансових даних, що потребують агрегування.

Предмет дослідження: Технології та інструменти реалізації процесів інтеграції даних і візуалізації результатів опрацьованих даних.

Мета роботи: Розробка автоматизованого інструмента для інтеграції фінансових даних з різних джерел з можливістю їх аналізу та візуалізації.

Завдання:

– аналіз технологій та інструментів для збору, трансформації та візуалізації даних;

– побудова архітектури інструменту для збору даних з фінансових джерел;

– реалізація функціоналу для збереження даних у базі даних з доступом через API;

– створення інтерфейсу користувача для візуалізації аналітичної інформації;

ОСНОВНІ РЕЗУЛЬТАТИ Розроблено повнофункціональний інструмент автоматизованого збору, трансформації та збереження фінансових даних із відкритих джерел. Реалізовано стабільний прикладний інтерфейс доступу до даних і вебінтерфейс для інтерактивної візуалізації результатів інтеграції.

Практичне значення дослідження: полягатиме у впровадженні розробленого інструменту для автоматизованого моніторингу фінансових активів в умовах фінансового прогнозування та систем підтримки прийняття рішень

Ключові слова інтеграція даних, багатошарова архітектура, ETL-процес, фінансові дані, автоматизація, API, візуалізація даних, Streamlit, FastAPI, PostgreSQL, Supabase.

**Зміст**

# ВСТУП

У сучасних умовах розвитку економіки одним із базових фактором успішного ведення бізнесу та прийняття управлінських рішень стає здатність ефективно працювати з даними. Інформація надходить з різноманітних джерел — бірж, банків, API фінансових сервісів, криптовалютних платформ і потребує об’єднання, обробки, збереження у необхідному для проведення її обробки вигляді що має допомогти у прийнятті зважених рішень. Саме тому актуальним є створення інструментів, здатних інтегрувати дані з різних форматів та джерел у єдиний аналітичний простір.

Метою кваліфікаційної роботи є розробка автоматизованого інструмента для інтеграції фінансових даних з різних джерел з подальшою можливістю їх аналізу та візуалізації. У рамках реалізації було створено веб-застосунок, який об’єднує API-запити, ETL (Extract, Transform, Load) - конвеєр, базу даних і клієнтський інтерфейс, що дозволяє працювати з даними про курси валют та акції.

Об’єктом дослідження виступають інформаційні потоки фінансових даних, що потребують агрегування. Предметом дослідження є технології та інструменти реалізації процесів інтеграції даних і візуалізації результатів опрацьованих даних.

Завдання роботи полягає в:

– аналізі технологій та інструментів для збору, трансформації та візуалізації даних;

– побудові архітектури інструменту для збору даних з фінансових джерел;

– реалізації функціоналу для збереження даних у базі даних з доступом через API;

– створенні інтерфейсу користувача для візуалізації аналітичної інформації;

Практична цінність роботи полягає у створенні зручного інструмента для відслідковування фінансової аналітики.

Методи дослідження включають системний аналіз, програмну реалізацію компонентів ETL-процесів, побудову API за допомогою FastAPI, застосування

бібліотек Python для аналітики та візуалізації і використання PostgreSQL як основного сховища.

# РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ІНТЕГРАЦІЇ ДАНИХ ДЛЯ АНАЛІЗУ ТА ВІЗУАЛІЗАЦІЇ

## 1.1. Роль інтеграції даних у сучасній науці та цифровій економіці

У сучасному світі спостерігається стрімке зростання обсягів інформації, що супроводжується ускладненням завдань її обробки, інтерпретації та використання. За даними аналітичного сервісу Rivery, обсяг створених і реплікованих даних у світі щороку стрімко зростає, і до 2025 року очікується досягнення межі у понад 181 зетабайт [1]. Це породжує потребу не лише у зберіганні даних, а й у їх ефективній інтеграції з різноманітних джерел з метою забезпечення цілісності, доступності та аналітичної корисності.

Інтеграція даних (Data Integration) — це процес об'єднання інформації з різних джерел у єдину узгоджену структуру, що включає етапи збирання, очищення, перетворення та збереження даних для подальшої обробки або аналізу [2] [3]. На своєму початку, інтеграція даних виникла у відповідь на потребу синхронізувати облікову інформацію з кількох відомчих систем, таких як бухгалтерія і склад, але на сьогодняшній день вона є основою бізнес-аналітики (BI), корпоративних інформаційних систем (ERP, CRM) і систем підтримки прийняття рішень (DSS) [3].

В епоху цифровізації інтеграція даних набуває стратегічного значення, особливо у контексті таких галузей як фінанси, охорона здоров’я, наука, логістика, кібербезпека тощо. Дані надходять у реальному часі з численних джерел — вебсайтів, сенсорних пристроїв, відкритих API, внутрішніх баз — і потребують об'єднання у спільне інформаційне поле. Як результат, виникає поняття «дані як актив» (data as an asset), згідно з яким дані вважаються стратегічним ресурсом, подібним до капіталу або людських ресурсів [4] [5].

Сучасні інформаційні системи орієнтовані на **сервісну модель**, де дані отримуються з численних незалежних джерел через API або стримингові канали. Це потребує гнучкої, масштабованої та стабільної архітектури інтеграції, що здатна адаптуватись до динамічності джерел, змін у структурі даних та високих вимог до якості інформації.

Інтеграція даних відіграє вирішальну роль у побудові **data-driven систем**, які спираються на автоматизовану аналітику, машинне навчання та предиктивне моделювання. Згідно з дослідженням McKinsey, організації, які ефективно використовують дані, можуть підвищити свою продуктивність до 20% [6].

Таким чином, інтеграція даних є не лише технічним процесом, але й методологічним принципом, що забезпечує побудову систем, орієнтованих на якісне управління інформаційними потоками. Вона є передумовою формування єдиної платформи знань, що необхідна для сучасних аналітичних і операційних рішень у більшості сфер людської діяльності.

### 1.2. Еволюція підходів до інтеграції даних

Історія розвитку інтеграції даних як окремої галузі нерозривно пов’язана з еволюцією інформаційних систем, розвитком комп’ютерних технологій та ускладненням викликів в сфері цифрового управління. Ще на початку 1960-х років проблемою інтеграції був процес, який вимагав ручного перенесення інформації або використання обмежених інтерфейсів, таких як магнітні стрічки чи фізичні накопичувачі [7] [8]. У цей період інтеграція сприймалась скоріше як технічна проблема перенесення даних, ніж як аналітична чи концептуальна задача.

Поява централізованих **реляційних баз даних (RDBMS)** у 1970–1980-х роках (Oracle, IBM DB2, Microsoft SQL Server) дала новий поштовх до інтеграції, оскільки дозволила створювати спільні сховища даних для різних

підрозділів компанії. Це стало основою для побудови **OLAP-систем (On-Line Analytical Processing)**. Ці системи використовували багатовимірні структури даних, відомі як "куби", що дозволяло зберігати великі обсяги інформації з різних джерел у спеціально спроєктованих багатовимірних структурах, що забезпечувало швидкий доступ до даних та можливість їх аналізу з різних перспектив, таких як час, географія чи категорії продуктів [9].

У 1990-х роках широкого поширення набуває концепція **Data Warehouse** — централізованого сховища, яке акумулює дані з оперативних систем, приводить їх до спільної структури та надає можливість виконання складних аналітичних запитів [10]. Засновник концепції, Білл Інмон, визначав, що "сховище даних — це предметно-орієнтована, інтегрована, незмінна та часово залежна колекція даних, що підтримує процес прийняття управлінських рішень" [11].

Разом із розвитком Data Warehouse виникає необхідність у формалізації процесу підготовки даних до зберігання, що призводить до поширення терміну **ETL** — процесу витягу, трансформації та завантаження даних. Цей підхід набуває популярності завдяки таким платформам, як Informatica, Talend, IBM DataStage та Microsoft SQL Server Integration Services (SSIS), які забезпечують ефективне збирання, очищення, консолідацію та збереження даних у спільну базу для подальшої звітності [12]. Основною ідеєю, закладеною в розвиток Data Warehouse було збирання даних з розрізнених джерел, їх очищення, консолідація та збереження у спільну базу для подальшої звітності.

У 2000-х роках, на тлі зростання обсягів даних (Big Data) та стрімкого розвитку інтернету, виникли нові вимоги до інтеграції даних [13]. Це призвело до появи підходу **ELT (Extract, Load, Transform)**, який "переносить обчислювальне навантаження на сховище даних, дозволяючи спочатку

завантажувати дані, а потім трансформувати їх безпосередньо в середовищі сховища" [14]. Паралельно почали активно застосовуватися **streaming-процеси** (наприклад, Apache Kafka, Apache Flink), де дані почали оброблятися безпосередньо під час надходження, що забезпечувало майже миттєву аналітику [13].

У 2010-х роках, із поширенням мікросервісної архітектури, обробки в хмарі та API-економіки, інтеграція даних перейшла на новий рівень, що призвело до активного розвитку інтеграційних платформ як сервісу (iPaaS), які забезпечують гнучке, масштабоване та автоматизоване об'єднання різних додатків і джерел даних [15]. До найпопулярніших рішень цього періоду належать Zapier, який дозволяє створювати інтеграції без написання коду [16], MuleSoft із орієнтацією на API-мережі [17], Apache NiFi як потужний інструмент потокової обробки даних та dbt Cloud для трансформації даних у хмарних сховищах [15], що спрощує створення конвеєрів обробки даних без класичного програмування. У паралель, **мови веб-програмування** (наприклад, Python з бібліотеками pandas, requests, SQLAlchemy) відкривають шлях до кастомізованих, модульних рішень, що стали актуальними у дослідницьких та стартап-орієнтованих середовищах.

Сьогодні інтеграція даних є динамічною галуззю, де поєднуються:

– **традиційні сховища** (PostgreSQL, BigQuery, Snowflake);

– **stream-платформи** (Kafka, Pulsar);

– **data orchestration** інструменти (Apache Airflow, Dagster);

– **ETL/ELT-бібліотеки** (Kedro, Prefect, Pandas, dbt);

– **візуальні low-code платформи** (Power BI, Data Studio, Tableau).

Із розвитком хмарних технологій зростає популярність рішень, орієнтованих на cerverless-архітектуру, де інтеграція виконується через події (event-driven processing), що дозволяє зменшити витрати та пришвидшити масштабування [18].

Отже, розвиток підходів до інтеграції даних пройшов шлях від простого копіювання файлів до складних багаторівневих конвеєрів у реальному часі, що спираються на гібридні архітектури, обчислення в хмарі та автоматизовану логіку трансформацій. Історична динаміка цієї галузі відображає зростаючу складність завдань, які стоять перед дослідниками, інженерами даних та розробниками сучасних інформаційних систем.

## 1.3. Сучасні технології та інструменти інтеграції даних

### 1.3.1 Програмні мови та середовища

У сучасному ландшафті інтеграційних рішень мова програмування Python виступає одним із найпоширеніших інструментів завдяки своїй простоті, гнучкості та потужній екосистемі бібліотек, які охоплюють повний спектр задач: від обробки даних і взаємодії з API до підключення до баз даних та побудови інтерактивних інтерфейсів. Зокрема, бібліотека pandas стала де-факто стандартом для обробки табличних даних, надаючи зручні структури даних та функції для маніпуляцій із числовими таблицями та часовими рядами. Для здійснення HTTP-запитів до зовнішніх API широко використовуються бібліотеки **requests** та **httpx**, а **SQLAlchemy** забезпечує ORM-доступ до реляційних баз даних, таких як PostgreSQL, спрощуючи взаємодію з базами даних через об'єктно-реляційне відображення.

Значною перевагою Python є підтримка як синхронного, так і асинхронного виконання, що дозволяє адаптувати ETL-процеси під навантаження та паралельну обробку. Крім того, Python органічно поєднується з мовою SQL, яка все ще залишається незамінною у випадках, коли трансформації виконуються безпосередньо в сховищі. У деяких системах, зокрема у хмарних аналітичних платформах, таких як Snowflake та BigQuery, SQL використовується не лише для запитів, а й для

управління моделями даних, що підкреслює важливість інтеграції Python із SQL у сучасних аналітичних рішеннях [19].

### 1.3.2 Інструменти для збирання і оркестрації даних

Процес інтеграції даних виходить далеко за межі простого отримання з API — він включає також оркестрацію, логіку повторного запуску, моніторинг, обробку помилок і залежностей [20]. У зв'язку з цим активно розвиваються інструменти управління робочими потоками (workflow orchestration), серед яких виділяються Apache Airflow, Prefect, Dagster. Apache Airflow, зокрема, дозволяє задавати графи залежностей між задачами, керувати середовищем виконання, зберігати логи та автоматизовувати запуск процесів за розкладом або за подією [20].

Інший важливий клас інструментів — це платформи ETL та ELT, серед яких Talend, Apache NiFi, Airbyte, Hevo. Вони надають користувачеві графічні інтерфейси для побудови конвеєрів даних, часто включаючи вбудовану підтримку понад сотні джерел — від Google Sheets і MySQL до Shopify чи Stripe [21] [22] [23]. З'являється також все більше low-code/no-code платформ, які орієнтовані на аналітиків або бізнес-користувачів, дозволяючи будувати інтеграції без глибоких знань програмування [21] [22].

### 1.3.3 Сховища даних

Після обробки та нормалізації дані потребують збереження у сховищі, яке забезпечує як швидкий доступ до інформації, так і можливість виконання аналітичних запитів. Вибір сховища залежить від масштабу, характеру даних та обраної архітектури.

Серед реляційних баз даних особливо виділяється PostgreSQL — потужне, відкрите рішення, що підтримує роботу з JSON через типи даних json та jsonb. Тип jsonb зберігає дані у бінарному форматі, що дозволяє ефективно індексувати та швидко обробляти запити до JSON-структур [24].

Крім того, PostgreSQL підтримує різноманітні типи індексів, такі як GIN та GiST, що забезпечують гнучку систему індексації та дозволяють масштабуватись через кластеризацію [24]. Серед альтернативних рішень для зберігання даних розглядаються різні варіанти залежно від поставлених вимог. Наприклад, MySQL демонструє меншу функціональність у контексті складної аналітики, що обмежує її використання для інтенсивних запитів. SQLite, своєю чергою, не підходить для багатокористувацького доступу через особливості архітектури та обмеження на паралельність операцій.

У сфері аналітики дедалі більше використовуються хмарні колоночні сховища, такі як Google BigQuery, Snowflake, Amazon Redshift. Ці платформи дозволяють обробляти великі об’єми даних завдяки оптимізації запитів на рівні колонок, автоматичному масштабуванню та тісній інтеграції з іншими хмарними сервісами. Наприклад, Google BigQuery пропонує серверлес-архітектуру з автоматичним розподілом ресурсів, що забезпечує ефективне виконання аналітичних запитів [25]. Snowflake використовує мікропартіційоване зберігання даних та розділення обчислювальних і зберігаючих ресурсів, що дозволяє досягти високої продуктивності та масштабованості [26]. Amazon Redshift, у свою чергу, використовує колоночну архітектуру з масовою паралельною обробкою, що оптимізує продуктивність запитів навіть при великих обсягах даних [26].

У проєктах, де потрібна швидка побудова MVP або демонстраційних інтерфейсів, зручним варіантом є Supabase — хмарна платформа на базі PostgreSQL, що надає вбудований REST API, автентифікацію, підписку на події (realtime) і просту адмінку без додаткового налаштування. Supabase автоматично генерує RESTful API з вашої схеми бази даних, дозволяючи швидко створювати інтерактивні додатки без необхідності писати бекенд-код. Крім того, платформа підтримує реальний час через

WebSocket-підписки, що дозволяє додаткам реагувати на зміни в базі даних миттєво [27].

### 1.3.4 API та сервіси як середовище взаємодії

У сучасному середовищі інтеграційних рішень більшість даних надходить не з файлів, а через API, де стандартом де-факто є RESTful API, які дозволяють здійснювати запити до сторонніх сервісів за допомогою стандартних HTTP-методів, таких як GET, POST, PUT і DELETE [28]. Для побудови RESTful-інтерфейсів у Python-розробці широке застосування отримав фреймворк FastAPI, що відзначається високою швидкодією завдяки використанню стандарту ASGI, підтримкою асинхронного виконання, а також автоматичною генерацією документації OpenAPI та JSON Schema без додаткових налаштувань. Крім FastAPI, популярними залишаються інші фреймворки, такі як Flask і Django у Python-екосистемі, Express для Node.js і Spring Boot для Java, що забезпечують широкі можливості для створення інтеграційних сервісів, здатних обробляти великі потоки даних і взаємодіяти з різноманітними джерелами інформації [28].

Якщо інтеграція потребує двосторонньої передачі даних у реальному часі або підвищеної ефективності в обробці великих обсягів інформації, дедалі активніше застосовуються альтернативні технології, такі як GraphQL та gRPC. GraphQL, розроблений Facebook, дозволяє клієнтам точно вказувати, які дані їм потрібні, що мінімізує обсяг переданої інформації та підвищує ефективність комунікації. У свою чергу, gRPC, створений Google, пропонує бінарний протокол для високошвидкісної взаємодії між сервісами та підтримує потокову передачу даних, що робить його особливо придатним для великих розподілених систем [29].

Окрім серверної інтеграції, важливим напрямом є взаємодія з кінцевим користувачем. Для створення візуальних інтерфейсів аналітики активно використовуються інструменти на Python, такі як Streamlit та Dash,

які дозволяють без великих витрат часу будувати інтерактивні веб-застосунки для роботи з даними у реальному часі [28]. Поряд із цим активно використовуються потужні BI-платформи, такі як Power BI, Tableau та Looker Studio, які надають можливість підключення до різноманітних джерел даних, включно з SQL-базами та API, і дозволяють формувати інтерактивні аналітичні панелі для оперативного ухвалення рішень [29].

Отже, ефективна реалізація інтеграційного рішення неможлива без глибокого розуміння екосистеми інструментів, що охоплюють повний цикл роботи з даними — від отримання через API та оркестрації потоків до зберігання в оптимізованих сховищах і надання доступу через гнучкі інтерфейси. Використання мови програмування Python разом із потужними бібліотеками для обробки, аналітики та побудови веб-інтерфейсів, а також сучасних платформ зберігання даних і BI-інструментів дозволяє створити масштабовану, автоматизовану систему для інтеграції різнорідних джерел. Це закладає технологічну основу для реалізації аналітичних платформ нового покоління, де ключовими є швидкість та автоматизація.

### 1.4. Проблеми, що залишаються відкритими у сфері інтеграції даних

Попри стрімкий розвиток технологій, процес інтеграції даних залишається однією з найскладніших задач в області інформаційних систем. Це пов’язано не лише з технічними викликами, а й із проблемами семантичної відповідності, надійності джерел, обмеженнями інфраструктури та браком стандартизації [30].

Однією з фундаментальних проблем є гетерогенність джерел. Дані можуть мати різні структури, формати представлення та семантичні інтерпретації. Навіть при використанні формально однакових структур даних часто спостерігається розбіжність у значеннях, одиницях

вимірювання або правилах агрегації, що унеможливлює просту консолідацію [31].

Наступною проблемою виступає якість даних (data quality). До 30% корпоративних рішень приймаються на основі неповних або спотворених даних. Джерела, які з технічної точки зору є доступними, не завжди забезпечують стабільність, актуальність чи коректну структуру відповідей. Пропуски, дублікати, аномальні значення, а також оновлення API без попередження — усе це знижує надійність автоматизованих ETL-конвеєрів [32].

Значним викликом залишається динаміка джерел. Багато платформ оновлюють API, обмежують кількість запитів, змінюють формати відповідей. Це потребує постійного моніторингу змін, адаптації скриптів і ризикує порушити цілісність аналітичного процесу. Більшість ETL-систем потребують щонайменше 20% ресурсу лише на обслуговування та оновлення інтеграцій [33].

Не менш складним завданням є інтеграція даних у реальному часі. Хоча існують потужні стримингові системи (Kafka, Flink, Pulsar), їх впровадження вимагає технічного досвіду, налаштування складної інфраструктури й часто не виправдане для середніх за масштабом проєктів. Навіть у великих компаніях перехід до real-time інтеграції часто стикається з опором через складність забезпечення надійності, черговості та повторюваності подій [34].

Суттєвою проблемою також є відсутність уніфікованих підходів до семантичної інтеграції. Хоча в літературі активно розробляються онтологічні моделі, що дозволяють зіставляти сутності на абстрактному рівні, у практиці застосування таких рішень зустрічається рідко — через складність реалізації, відсутність готових інструментів і низький рівень стандартизації в індустрії [35].

З організаційного боку існує проблема доступності та відкритості джерел. Обмеження доступу, нестабільність хостингів, недостатня документація або ліцензійні обмеження часто стають реальними бар’єрами для створення стабільних інтеграцій. Це особливо стосується міжорганізаційної або міждержавної взаємодії [36].

Нарешті, недостатньо вивченим залишається питання оцінки ефективності інтеграції. У більшості випадків ефективність вимірюється лише часовими або технічними метриками (час оновлення, швидкість запиту), тоді як такі аспекти, як гнучкість, точність зіставлення чи підтримка змін у схемах, залишаються поза увагою [37].

Таким чином, незважаючи на значний прогрес у розвитку технологій інтеграції, ця сфера зіштовхується як з технічними, так і концептуальними викликами. Гетерогенність джерел, нестабільність API, проблеми якості та реального часу, відсутність семантичної уніфікації і труднощі з доступом до даних формують багатовимірну зону ризику, яка вимагає постійного технічного супроводу, гнучких архітектурних рішень і посиленого методологічного підходу.

# РОЗДІЛ 2. АНАЛІТИЧНИЙ ОГЛЯД І ПРОЄКТУВАННЯ ІНСТРУМЕНТУ

## 2.1. Постановка задачі проєктування інструменту

На основі аналізу наукових джерел та огляду сучасних практик в галузі обробки фінансових даних було встановлено, що інтеграція даних залишається актуальною й водночас складною проблемою. Виклики виникають як на рівні технічної реалізації (різнорідність структур API, нестабільність відповідей, різні часові формати), так і на рівні організації процесу обробки (неузгодженість типів, відсутність гарантованої якості даних, дублікати). Окремою проблемою є необхідність забезпечення зручного доступу до вже оброблених даних у формі, придатній для візуалізації або подальшого аналізу.

Виявлені проблеми зумовлюють потребу у створенні гнучкого інструмента, здатного здійснювати інтеграцію фінансових даних із різних джерел, обробляти ці дані відповідно до єдиної внутрішньої моделі та забезпечувати їх доступність через зручний інтерфейс. При цьому дослідження має охоплювати не лише побудову архітектури такого інструмента, але й оцінку можливості використання конкретних підходів до збору, трансформації, зберігання та представлення даних у контексті відкритих джерел.

У цьому контексті сформульовано такі дослідницькі задачі:

- Виявити та проаналізувати відкриті джерела фінансових даних, що придатні для інтеграції.
- Визначити вимоги до трансформації отриманих даних, їх збереження з використанням реляційної моделі бази даних.
- Обрати архітектурний підхід до організації доступу до оброблених даних, проаналізувати можливості реалізації запитів, фільтрації та оновлення записів.

- Визначити вимоги до клієнтського інтерфейсу для перегляду фінансових даних, оцінити доцільність використання інструментів інтерактивної візуалізації, критерії зручності та доступності для користувача.

Кожна задача передбачає перевірку певного технічного або концептуального припущення — зокрема, що застосування асинхронного підходу дозволить зменшити час збору великої кількості валютних даних, а уніфікація структури даних зменшить кількість помилок при обробці. Формулювання задач у рамках дослідження забезпечує перехід від проблемної області до структурованої моделі рішення, яка може бути оцінена за критеріями якості.

У зв’язку з цим, критеріями успішності реалізації інструмента визначено: реалізацію функціонального циклу отримання, обробки, зберігання та представлення фінансових даних; відповідність структури API та бази даних вимогам доступності й цілісності; забезпечення стабільної роботи інтерфейсу користувача; логічну та форматну узгодженість оброблених даних. Ці аспекти створюють підґрунтя для подальшої оцінки ефективності реалізованого рішення у практичному розділі.

## 2.2. Обґрунтування підходу до архітектури інструмента

Вибір архітектурного підходу до побудови інструмента інтеграції фінансових даних зумовлюється характером поставлених задач, різнорідністю джерел даних і потребою в масштабованості та модульності. На основі аналізу сучасних концепцій побудови інформаційних систем було обґрунтовано доцільність використання багатошарової архітектури як найбільш відповідної для створення дослідницького інструмента в сфері обробки фінансової інформації.

Багатошарова архітектура передбачає розподіл системи на окремі рівні: збір даних, обробка, зберігання, доступ і подання. Такий поділ забезпечує логічну ізоляцію функцій, дозволяє незалежно модифікувати компоненти та полегшує підтримку інструмента. У контексті дослідження кожен рівень виконує чітко визначену роль, і їх взаємодія забезпечує повний цикл роботи з даними: від моменту отримання інформації з відкритих джерел до виведення її у зручному форматі для користувача.

На рівні обробки даних доцільно застосовано модель ETL (Extract, Transform, Load), яка дозволяє структуровано організувати процес інтеграції. Ця модель була обрана через її гнучкість: вона дає змогу чітко відокремити логіку збору, трансформації та зберігання даних, що особливо важливо при роботі з джерелами, які мають різні формати, структури та вимоги до частоти оновлення. Застосування ETL у межах багатошарової архітектури дозволяє локалізувати складність і забезпечити масштабованість системи в майбутньому.

Ключовим чинником при виборі такої архітектури була також потреба в модульності. Це означає, що кожен компонент інструмента — незалежно від того, чи йдеться про завантаження даних, трансформацію, чи взаємодію з базою — має бути реалізований як окремий модуль, який може бути протестований і замінений окремо від решти. Такий підхід дозволяє знизити зв’язність між компонентами, прискорити розробку та забезпечити можливість розширення функціоналу без суттєвих змін у загальній структурі.

У процесі теоретичного обґрунтування розглядалися й альтернативні архітектурні рішення, зокрема монолітний підхід та варіанти з використанням потокових моделей обробки. Проте вони не забезпечували належного рівня керованості та адаптивності до змін, які можуть виникати в реальних джерелах даних (API, формати, частота оновлення).

Багатошарова архітектура в поєднанні з ETL дозволяє врахувати ці фактори й мінімізувати технічні ризики.

Загалом, запропонований підхід до архітектури інструмента обґрунтований з урахуванням сучасних вимог до систем інтеграції даних, характеру дослідження та перспектив подальшої модифікації й масштабування рішення.

### 2.3. Вибір і обґрунтування технологій та інструментів

Для побудови ефективного інструмента інтеграції фінансових даних постало завдання вибору відповідних технологій і засобів реалізації, що мали забезпечити відповідність таким критеріям, як гнучкість, масштабованість, інтеграційна сумісність та підтримка відкритих стандартів. Рішення приймалися на основі аналізу сучасного технологічного ландшафту, представленого у пункті 1.3, із фокусом на концептуальні переваги кожного з можливих підходів.

На рівні серверного інтерфейсу взаємодії з даними було обрано FastAPI серед різноманітних фреймворків для побудови RESTful API. Порівняно з альтернативами, FastAPI забезпечує високу продуктивність завдяки використанню стандарту ASGI, вбудовану підтримку асинхронного виконання запитів, автоматичну генерацію документації OpenAPI та інтеграцію з типізацією на основі Python. Ці особливості дозволяють значно пришвидшити розробку, підвищити надійність коду, полегшити інтеграцію з іншими сервісами та забезпечити масштабованість рішення.

У якості інструмента для зберігання даних було обрано реляційну СУБД PostgreSQL. Вона має багаторічну репутацію надійного рішення з підтримкою складних запитів, розширеної типізації, JSONB-полів, різних стратегій індексації (GIN, GiST), а також транзакційної узгодженості (ACID). Платформа Supabase, побудована на PostgreSQL, обрана як сервіс для

швидкого розгортання із вбудованим REST API, WebSocket-підтримкою та механізмами авторизації, що дозволяє уникнути додаткових витрат на конфігурацію та адміністрування.

Щодо інструментів клієнтського рівня, обрано Streamlit — Python-фреймворк для швидкого створення інтерактивних вебінтерфейсів. У порівнянні з альтернативами (Dash, Panel або повноцінні фреймворки на JavaScript) Streamlit пропонує найкоротший цикл розробки MVP для Python-орієнтованих рішень. Dash забезпечує гнучкішу побудову динамічних елементів, але потребує складнішої архітектури і більше конфігурації. Streamlit дозволяє будувати користувацький інтерфейс з мінімальним кодом, інтегруючи його безпосередньо з аналітичними обчисленнями на Python.

На рівні бібліотек було прийнято рішення використовувати pandas (для табличної обробки), SQLAlchemy (ORM-зв’язок із базою), yfinance (отримання фінансових даних з Yahoo Finance), та httpx (асинхронні HTTP-запити до API НБУ). Розглядались також альтернативи, такі як psycopg2 для прямої роботи з PostgreSQL (менш гнучкий в порівнянні з ORM), або requests (не підтримує асинхронність). Обрані бібліотеки дозволили забезпечити прозору обробку даних і простоту розширення коду.

Таким чином, вибір технологій був зумовлений не лише технічною сумісністю, а й прагненням забезпечити прозорість, гнучкість, адаптивність та можливість подальшого масштабування проєкту. Усі компоненти інтегруються в єдину архітектуру, що відповідає цілям дослідження та дозволяє реалізувати повний цикл обробки фінансових даних: від отримання до візуалізації.

## 2.4. Вибір джерел даних

Питання вибору джерел даних є критично важливим елементом у дослідженні, що стосується побудови інструмента для інтеграції фінансових даних. Питання достовірності таких джерел охоплює низку аспектів, таких як надійність джерел, так і відповідність обраної методики цільовим завданням аналізу, а також загальну узгодженість та логічну цілісність отриманих даних.

У процесі порівняльного аналізу було розглянуто декілька відкритих API для отримання валютних курсів і біржових котирувань. Зокрема, API open.er-api.com демонструє зручність використання без авторизації, проте обмеження щодо частоти оновлень робить його непридатним для сценаріїв, де важлива оперативність. Exchangerate.host також розглядався як альтернатива, однак його доступність і відповідність офіційним джерелам виявилася нижчою порівняно з національними банками. У підсумку було віддано перевагу API НБУ, який забезпечує стабільний доступ до офіційних курсів у стандартизованому форматі.

Щодо джерел біржових даних, порівнювались Yahoo Finance та Alpha Vantage. Попри функціональну насиченість Alpha Vantage (наявність технічних індикаторів), його обмеження за кількістю запитів і необхідність API-ключів знизили ефективність при масштабованих сценаріях. Натомість Yahoo Finance, доступний через бібліотеку yfinance, не вимагає ключів, має хорошу стабільність і широке охоплення активів, що зробило його придатним для задач інтеграції без зайвого адміністративного навантаження.

Для забезпечення точної та продуктивної інтеграції важливо враховувати характеристики способу доступу до джерел. Наприклад, системи, які дозволяють гнучко керувати частотою запитів, швидко реагувати на затримки й повторно ініціювати запити, мають вищу надійність

в умовах нестабільного з'єднання. Саме тому перевага надається рішенням, які підтримують багатопоточну або асинхронну логіку обробки — вони здатні покращити загальну стабільність збору інформації. Такі принципи закладають підґрунтя для подальшої оптимізації продуктивності, що буде предметом розгляду в наступному розділі.

Окремим фактором є перевірка внутрішньої узгодженості даних. Це стосується збереження хронологічної послідовності, уникнення дублікатів, контролю за типами значень і відсутністю пропущених критичних полів. Створення уніфікованої схеми з чітким визначенням обов’язкових атрибутів дозволяє зменшити ризик помилок та підвищити якість подальшої обробки.

Отже, вибір джерел даних було зроблено на API НБУ та Yahoo Finance, які є оптимальними завдяки їхній структурованій організації та стабільній доступності. API НБУ забезпечує стандартизований JSON-формат валютних курсів, а Yahoo Finance пропонує широкий спектр біржових котирувань без потреби в авторизації. Такий вибір джерел оптимізує процеси доступу та обробки даних, забезпечуючи ефективну інтеграцію в розроблений інструмент.

# РОЗДІЛ 3. РЕАЛІЗАЦІЯ АВТОМАТИЗОВАНОГО ІНСТРУМЕНТА ДЛЯ ІНТЕГРАЦІЇ ТА АНАЛІЗУ ФІНАНСОВИХ ДАНИХ

## 3.1. Архітектура та загальна структура інструмента

Автоматизований інструмент для інтеграції фінансових даних, розроблений у межах даної роботи, побудований за принципами багатошарової архітектури. Така структура передбачає розподіл функціональності між незалежними компонентами, кожен з яких відповідає за окремий аспект системи — від збору даних і їхньої обробки до подання кінцевому користувачу. Це забезпечує гнучкість, масштабованість та зручність супроводу проєкту.

Загальна архітектура інструмента включає чотири шари: Presentation Layer, Application Layer, Data Processing Layer та Database Layer. Кожен із них реалізує окремий функціональний блок у загальній логіці роботи системи.

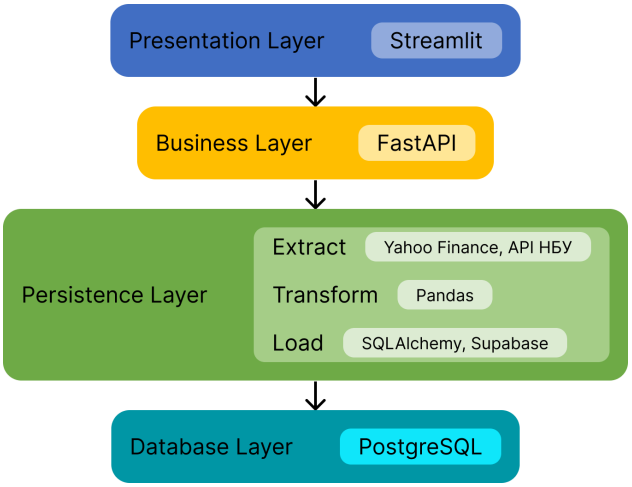


*Рисунок 3.1.1 – Багатошарова архітектура інструмента*

**Presentation Layer** відповідає за подання інформації у зручній для користувача формі. Цей шар реалізовано з використанням фреймворку Streamlit, який дозволяє створювати інтерактивні вебінтерфейси без потреби у фронтенд-розробці. Користувач має змогу обрати актив, вказати часовий період, переглянути графік динаміки цін або курсу валют,

ініціювати оновлення даних, а також експортувати результати у файл. Інтерфейс працює у режимі реального часу, надсилаючи запити до внутрішнього API.

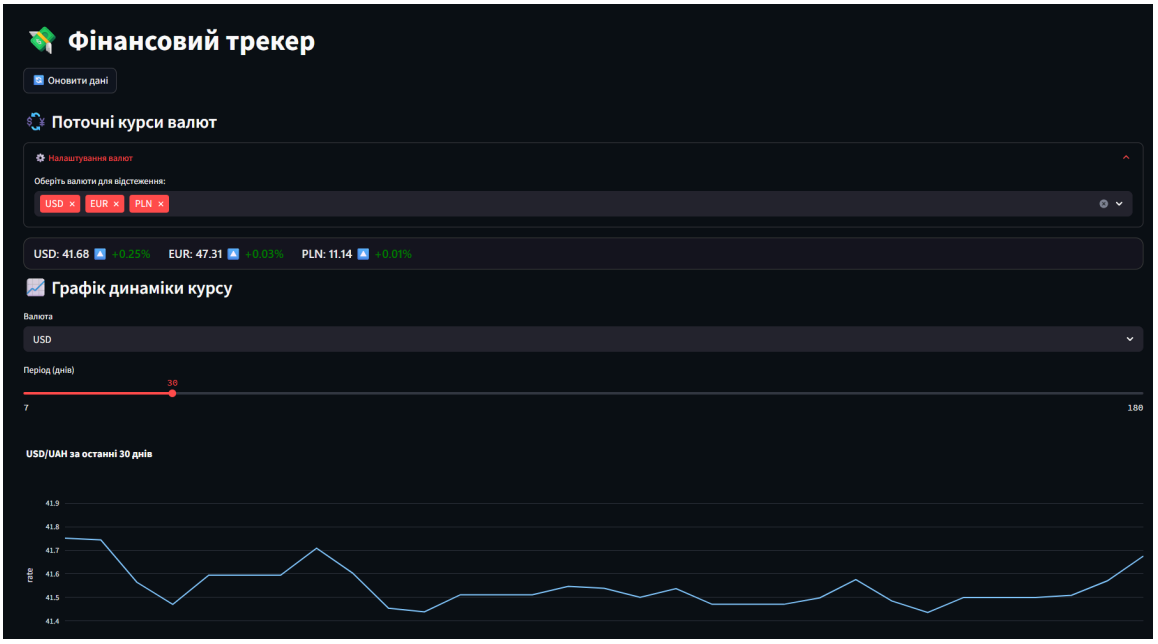


*Рисунок 3.1.2 – Фрагмент інтерфейсу інструмента на базі Streamlit*

**Application Layer** реалізує прикладну логіку взаємодії між клієнтською частиною та даними. Цей компонент побудовано за допомогою фреймворку FastAPI, що забезпечує обробку HTTP-запитів, валідацію параметрів через Pydantic-моделі, а також повернення структурованих відповідей у форматі JSON. FastAPI також автоматично генерує документацію у форматі OpenAPI, що полегшує тестування та інтеграцію. До основних ендпоінтів належать /exchange_rates, /assets, /stock_prices/{symbol}.


*Рисунок 3.1.3 – Документація API, згенерована FastAPI через Swagger UI*

**Data Processing Layer** відповідає за логіку обробки даних. У структурі інструмента цей рівень реалізує класичний ETL-підхід — витяг (Extract), трансформацію (Transform) та завантаження (Load) даних. На етапі витягу для кожного типу активів використовуються окремі джерела: для валют — API НБУ, для акцій — Yahoo Finance. Також у проєкті передбачена можливість обробки відповіді від API за допомогою бібліотеки requests, що дозволяє гнучко взаємодіяти із зовнішніми вебсервісами.

Трансформація даних здійснюється за допомогою бібліотеки pandas: дані нормалізуються, уніфікуються часові мітки, здійснюється типізація числових полів, сортування за датами та фільтрація порожніх значень. Після цього дані передаються для збереження у базі даних через ORM-рівень (SQLAlchemy), що реалізує завантаження даних до відповідних таблиць.

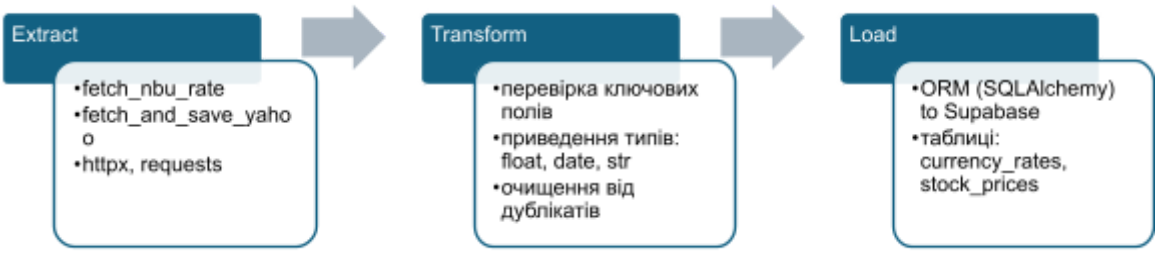


*Рисунок 3.1.4 – Логіка обробки даних в ETL-модулі*

**Database Layer** відповідає за довготривале зберігання фінансової інформації. У рамках реалізації інструмента використано PostgreSQL як основну СУБД, яка забезпечує збереження структурованих записів (курси валют, ціни акцій) із використанням типів Date, Float, String, JSON. Завдяки ORM-інтеграції вся взаємодія з базою здійснюється через SQLAlchemy, що забезпечує зручне та безпечне додавання, оновлення і видалення записів.

Базу даних розгорнуто у хмарному середовищі Supabase. Це дає змогу використовувати вбудовані REST-інтерфейси, механізми авторизації, а також функції реального часу. Таким чином, обраний підхід забезпечує простоту розгортання та масштабування інструмента без необхідності ручного налаштування серверної інфраструктури.

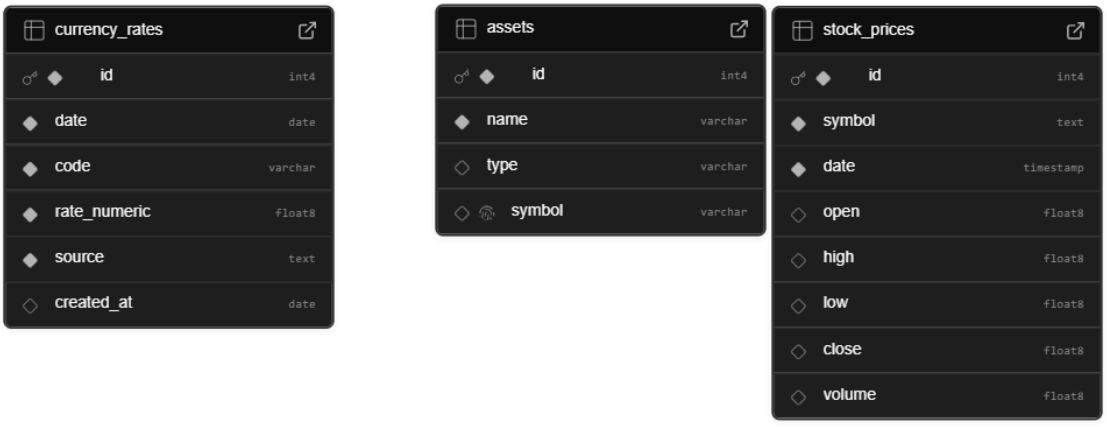


*Рисунок 3.1.5 – Структура таблиць у базі даних Supabase*

Отже, побудова архітектури автоматизованого інструмента для інтеграції фінансових даних здійснювалася з урахуванням принципів

багатошаровості, модульності та масштабованості. Кожен із визначених шарів — Presentation Layer, Application Layer, Data Processing Layer та Database Layer — виконує чітко визначену функцію та взаємодіє з іншими через стандартизовані інтерфейси. Така структура дозволяє забезпечити стабільність, гнучкість та подальший розвиток системи без необхідності її повної перебудови. Обрана архітектурна модель стала фундаментом для реалізації функціонального інструмента, що поєднує інтеграцію з відкритими джерелами, обробку фінансових даних і зручне представлення результатів кінцевому користувачу.

### 3.2 Реалізація ETL-процесів

Однією з центральних функцій інструмента є повноцінна реалізація ETL-процесів (Extract, Transform, Load), які забезпечують автоматичне отримання, обробку та збереження фінансових даних у базі. Кожен із етапів реалізовано у межах окремих модулів проєкту, що відповідає принципам модульності, масштабованості та відокремлення відповідальностей. У цьому підрозділі послідовно розглядаються технічні рішення, реалізовані для кожної з фаз — починаючи зі збору даних із відкритих API.

#### 3.2.1. Збір даних через API

Збір фінансових даних у розробленому інструменті реалізовано як перший етап класичного ETL-процесу. Джерелами інформації виступають офіційний API Національного банку України для валютних курсів та сервіс Yahoo Finance для котирувань акцій. Архітектурно ці компоненти ізольовано у вигляді модулів, кожен з яких відповідає за отримання та первинну перевірку даних.

Валютні курси отримуються через функцію fetch_nbu_rate_from_api(code, target_date), яка формує запит до відкритого ендпоінту НБУ. Для взаємодії із зовнішнім сервером використовується

асинхронна бібліотека httpx, що забезпечує неконфліктну інтеграцію у загальну архітектуру. Відповідь API перевіряється на статус, цілісність структури та наявність курсу для заданої дати. Якщо курс знайдено, він округлюється та передається до функції save_nbu_rate, яка перевіряє наявність запису в базі та зберігає новий лише за відсутності дубліката.

```python
async def fetch_nbu_rate_from_api(code: str, target_date: date) -> float | None:
    """Асинхронний запит до API НБУ для отримання курсу на задану дату."""
    url = NBU_URL.format(code.upper(), target_date.strftime('%Y%m%d'))
    async with httpx.AsyncClient() as client:
        response = await client.get(url)
        if response.status_code != 200:
            return None
        data = response.json()
        if not data:
            return None
        return round(float(data[0]["rate"]), 4)
```

*Рисунок 3.2.1.1 – Фрагмент асинхронного запиту до API НБУ*

Котирування акцій і криптовалют збираються за допомогою модулю data_sources/fetch_yahoo.py. Для цього використовується класична трирівнева реалізація: extract_yahoo_data(symbol, days) звертається до Yahoo Finance, отримуючи історичні дані у JSON-форматі.

```python
def extract_yahoo_data(symbol, days):
    url = f"https://query1.finance.yahoo.com/v8/finance/chart/{symbol}?interval=1d&range={days}d"
    headers = {"User-Agent": "Mozilla/5.0"}
    response = requests.get(url, headers=headers)
    data = response.json()

    if "chart" not in data or not data["chart"]["result"]:
        log_message(f"Дані не знайдені для символу: {symbol}")
        return None

    return data["chart"]["result"][0]
```

*Рисунок 3.2.1.2 – Побудова DataFrame з котирувань Yahoo Finance*

Усі запити до Yahoo виконуються синхронно, оскільки API не має обмежень швидкості в межах проєкту. При цьому обробка даних супроводжується перевірками на наявність timestamp, символів та типових полів (open, close, volume), що дозволяє уникати логічних помилок і

дублювання. У разі помилки логіка не переривається — інформація передається у файл журналу для подальшого аналізу.

Такий підхід забезпечує надійний і контрольований механізм отримання первинних фінансових даних, які уніфікуються на подальших етапах обробки.

### 3.2.2. Трансформація та валідація даних

Другим етапом ETL-процесу після збору є трансформація та валідація отриманих даних. Цей етап передбачає приведення сирих даних з відкритих API до уніфікованого табличного формату, перевірку на логічну узгодженість, відповідність типів та відсутність дублювань. У розробленому інструменті трансформація реалізована окремо для кожного джерела — Yahoo Finance (котирування акцій) та API НБУ (валютні курси).

У випадку Yahoo Finance, сирі дані надходять у вигляді вкладеного JSON. Функція transform_yahoo_data виділяє часові мітки (timestamp) та поля з цінами (open, high, low, close, volume) і формує з них структуру типу pandas.DataFrame. Одночасно здійснюється перетворення timestamp у формат datetime, додавання поля symbol і впорядкування стовпців для уніфікованої структури даних.

```
def transform_yahoo_data(result, symbol):
    timestamps = result["timestamp"]
    indicators = result["indicators"]["quote"][0]

    df = pd.DataFrame(indicators)
    df["date"] = pd.to_datetime(timestamps, unit="s")
    df["symbol"] = symbol
    df = df[["symbol", "date", "open", "high", "low", "close", "volume"]]
    return df
```

*Рисунок 3.2.2.1 – Формування структури DataFrame з JSON-даних Yahoo Finance*

Далі відбувається попередній аналіз якості даних: виведення статистики, перевірка наявності пропущених значень, діагностика розподілу цін. Функція fetch_and_save_yahoo також включає ряд перевірок, що дозволяють виявити аномалії:

- усі значення low мають бути меншими або рівними high;
- значення open і close мають знаходитися у межах діапазону low – high;
- перевіряється наявність дублікатів за парою symbol + date.

```
# Перевірки
condition1 = (df["low"] <= df["high"]).all()
condition2 = ((df["open"] >= df["low"]) & (df["open"] <= df["high"])).all()
condition3 = ((df["close"] >= df["low"]) & (df["close"] <= df["high"])).all()
duplicates = df.duplicated(subset=["symbol", "date"]).sum()
```

*Рисунок 3.2.2.2 – Логічні перевірки цінових значень і дублікатів*

У випадку з валютами трансформація є суттєво простішою — оскільки відповідь API НБУ містить лише одне числове значення rate. Операція округлення до чотирьох знаків після коми здійснюється безпосередньо в межах функції fetch_nbu_rate_from_api, а валідація та підготовка до збереження — у функції save_nbu_rate. Саме там перевіряється наявність запису за комбінацією date + code + source, і новий запис створюється лише за відсутності дубліката.

```
existing = db.query(CurrencyRate).filter(
    and_(
        CurrencyRate.date == target_date,
        CurrencyRate.code == code.upper(),
        CurrencyRate.source == "NBU"
    )
).first()

if existing:
    return existing.rate_numeric
```

*Рисунок 3.2.2.3 – Перевірка унікальності запису перед збереженням*

Таким чином, трансформація та валідація в інструменті реалізовані з урахуванням специфіки кожного джерела даних. Для складних структур (як у випадку з Yahoo) застосовується багаторівнева обробка з використанням pandas, а для лаконічних відповідей (як у API НБУ) — інтегрована логіка перевірки та збереження. Обидва підходи забезпечують надійність, цілісність і консистентність даних у базі, що є критично важливим для подальшого аналізу й візуалізації.

### 3.2.3. Збереження даних у базі даних

Заключним етапом ETL-процесу є збереження трансформованих і валідованих даних у базу даних. У межах розробленого інструмента цей етап реалізовано з використанням об'єктно-реляційного підходу (ORM), що забезпечує зручність, безпечність і контроль над структурою збережених даних. Як сховище використано PostgreSQL, розгорнуте у хмарному середовищі Supabase.

Збереження котирувань акцій і криптовалют реалізовано через функцію load_stock_prices, яка виконує підключення до бази даних, видаляє попередні записи для заданого активу (на основі поля symbol) і завантажує нові дані у таблицю stock_prices. Перед збереженням дані представлені у вигляді pandas.DataFrame.

```
def load_stock_prices(df):
    engine = create_engine(DB_CONFIG["url"])
    with engine.connect() as conn:
        conn.execute(text("DELETE FROM stock_prices WHERE symbol = :symbol"), {"symbol": df['symbol'].iloc[0]})
        df.to_sql("stock_prices", con=engine, if_exists="append", index=False)
```

*Рисунок 3.2.3.1 – Збереження оновлених котирувань акцій у таблицю stock_prices*

Така схема дозволяє завжди працювати з актуальним набором історичних даних без ризику накопичення застарілих або дубльованих записів. Видалення виконується на рівні SQL-команди DELETE, а запис — через метод to_sql, який інтегрується з SQLAlchemy.

У випадку валютного модуля, збереження здійснюється через ORM-модель CurrencyRate, визначену у файлі api/models.py. Функція save_nbu_rate створює новий екземпляр моделі лише у випадку, якщо запис за комбінацією date, code і source відсутній у базі:

```
record = CurrencyRate(
    date=target_date,
    code=code.upper(),
    rate_numeric=rate,
    source="NBU"
)
db.add(record)
db.commit()
```

*Рисунок 3.2.3.2 – Додавання нового валютного запису до таблиці currency_rates через ORM*

Для підключення до бази даних використовуються параметри, збережені у модулі config/settings.py, що дозволяє централізовано управляти середовищами та дотримуватись принципів безпечного зберігання облікових даних.

У таблицях передбачено первинні ключі, індекси та унікальні обмеження. Наприклад, для currency_rates встановлено обмеження UniqueConstraint("date", "code", "source"), що захищає від дублювання при асинхронному зборі історичних курсів.

Таким чином, збереження даних реалізовано із застосуванням сучасних практик обробки баз даних: використано ORM для безпечної взаємодії з моделями, забезпечено очищення попередніх записів перед завантаженням, дотримано структурної узгодженості таблиць, що дає змогу гарантувати цілісність і консистентність даних у базі на кожному етапі роботи інструмента.

### 3.3. Організація бази даних

Для забезпечення зберігання фінансової інформації в реалізованому інструменті використано реляційну базу даних PostgreSQL, розгорнуту у хмарному середовищі Supabase. Це рішення дозволяє поєднати переваги класичної SQL-моделі з сучасними інструментами REST-доступу, автентифікації, автоматичного логування та масштабованості.

У структурі бази даних визначено кілька основних таблиць, кожна з яких виконує чітко окреслену функцію в загальній логіці обробки та аналізу даних:

- currency_rates — зберігає курси валют за датами;
- stock_prices — зберігає історичні котирування акцій і криптовалют;
- assets — містить довідник відстежуваних фінансових активів.

Таблиця currency_rates містить поля date, code, rate_numeric, source, що дозволяє зберігати час, код валюти, числове значення курсу та джерело отримання. Первинним ключем виступає поле id, а для захисту від дублювання даних під час асинхронного збору реалізовано обмеження унікальності:

```
constraint uix_currency_date_code_source unique (date, code, source)
```

*Рисунок 3.3.1 – Фрагмент коду захисту від дублювання*

Також у цій таблиці створено індекси для прискорення пошуку за полями date, code, id, що оптимізує фільтрацію при запитах за часовим діапазоном або кодом валюти. Поле created_at ініціалізується автоматично значенням поточної дати.

| id int4 | date date | code varchar | rate_numeric float8 | source text | created_at date |
|---|---|---|---|---|---|
| 1 | 2025-04-17 | USD | 41.2152 | NBU | 2025-04-17 |
| 2 | 2025-04-17 | EUR | 46.8205 | NBU | 2025-04-17 |
| 4 | 2025-04-16 | USD | 41.1753 | NBU | 2025-04-17 |
| 5 | 2025-04-15 | USD | 41.3153 | NBU | 2025-04-17 |
| 6 | 2025-04-14 | USD | 41.3879 | NBU | 2025-04-17 |
| 7 | 2025-04-13 | USD | 41.4031 | NBU | 2025-04-17 |
| 8 | 2025-04-12 | USD | 41.4031 | NBU | 2025-04-17 |

*Рисунок 3.3.2 – Вигляд таблиці currency_rates*

Таблиця stock_prices зберігає біржові котирування активів з полями: symbol, date, open, high, low, close, volume. Тут первинним ключем виступає пара полів symbol + date, що відображає логіку унікальності для кожного запису в часовому ряді.

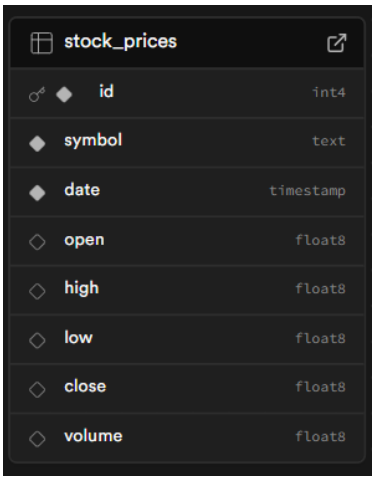


*Рисунок 3.3.3 – Структура таблиці stock_prices із використанням складеного ключа*

Дані до цієї таблиці надходять з модуля fetch_yahoo.py після трансформації у формат DataFrame. Завдяки суворій типізації (double precision, timestamp) забезпечується висока точність і коректна обробка історичних даних.

Таблиця assets виступає як центральний реєстр фінансових активів. Кожен запис має id, name, type (валюта, акція або криптовалюта) та symbol, що використовується для отримання зовнішніх даних. Крім первинного ключа, реалізовано перевірку цілісності значень через обмеження.

```
constraint assets_type_check check (
  (
    (type)::text = any (
      array[
        ('currency'::character varying)::text,
        ('stock'::character varying)::text,
        ('crypto'::character varying)::text
      ]
    )
  )
)
```

*Рисунок 3.3.4 – Обмеження в таблиці assets, що регулюють типи активів і унікальність символів*

Усі таблиці належать до схеми public, що дозволяє забезпечити сумісність із хмарною інфраструктурою Supabase без додаткового конфігурування.

Загалом структура бази даних розроблена з урахуванням модульності, нормалізації та продуктивності. Первинні ключі та унікальні обмеження гарантують цілісність даних, індекси прискорюють пошук, а типізація дозволяє точно зберігати фінансові показники. Такий підхід забезпечує надійну основу для подальшої аналітики, інтеграції та розширення системи.

### 3.4. Реалізація прикладного API

У межах реалізованого інструмента важливу роль відіграє прикладний програмний інтерфейс (API), що забезпечує взаємодію між зовнішнім вебінтерфейсом, базою даних та модулями збору фінансової інформації. API побудовано з використанням фреймворку **FastAPI**, який підтримує автоматичну генерацію документації, асинхронну обробку запитів та сувору типізацію параметрів через моделі Pydantic.

Файл main.py містить повну реалізацію API-інтерфейсу. На початку ініціалізується об’єкт FastAPI, а також створюється об’єкт engine для підключення до бази даних через SQLAlchemy:

```python
engine = create_engine(DB_CONFIG["url"])
SessionLocal = sessionmaker(bind=engine)

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

*Рисунок 3.4.1 – Підключення до бази даних та генерація сесії через SQLAlchemy*

Для перевірки роботи сервера реалізовано службові маршрути / і /ping, які повертають відповідні повідомлення. Основний функціонал API зосереджено навколо роботи з активами, історією котирувань та валютними курсами.

Запити до /assets дозволяють отримати перелік активів з бази (GET), додати новий запис (POST) або видалити актив за його ідентифікатором (DELETE). У разі додавання нового активу з типом stock або crypto ініціюється одразу збір історичних котирувань:

```python
if asset.type in ("stock", "crypto") and asset.symbol:
    success = fetch_and_save_yahoo(asset.symbol)
```

*Рисунок 3.4.2 – Умова для ініціалізації збору даних після додавання активу*

Для читання історії цін використано маршрут /stock_prices/{symbol}, який повертає впорядковані дані з таблиці stock_prices за допомогою ORM-запиту. Для маршруту передбачено підключення до бази через механізм Depends(get_db), що забезпечує безпечне використання сесії SQLAlchemy.

```
try:
    rows = db.query(StockPrice).filter(StockPrice.symbol == symbol).order_by(StockPrice.date).all()
    return [
        {
            "symbol": r.symbol,
            "date": r.date,
            "open": r.open,
            "high": r.high,
            "low": r.low,
            "close": r.close,
            "volume": r.volume
        }
        for r in rows
    ]
```

*Рисунок 3.4.3 – Отримання впорядкованої історії котирувань через ORM-модель StockPrice*

Маршрут /assets/update дозволяє перевірити наявність свіжих даних по кожному активу, і, у разі їх відсутності, автоматично ініціює повторний запит на завантаження. Перевірка виконується для кожного запису в таблиці assets, де symbol IS NOT NULL.

```
query = text("""
    SELECT 1 FROM stock_prices
    WHERE symbol = :symbol AND DATE(date) = :today
    LIMIT 1
""")
```

*Рисунок 3.4.4 – SQL-запит для перевірки наявності котирувань за поточну дату*

Для оновлення валют реалізовано окремий маршрут /currency/update, який запускає асинхронну функцію fetch_history для кожного з попередньо визначених валютних кодів. Завдяки використанню asyncio.run() забезпечується послідовне, але асинхронне завантаження даних з API НБУ:

```
async def run_all():
    for code in currencies_to_add:
        await fetch_history(db, code, 35)
```

*Рисунок 3.4.5 – Циклічне оновлення курсів валют у маршруті /currency/update*

Крім того, API підтримує параметризовані запити. Зокрема, у маршруті /currency/{code} користувач може передати start та end як

параметри через Query, після чого повертається відсортований список курсів валюти за вказаний період:

```
def get_currency_history(code: str,
                         start: date = Query(...),
                         end: date = Query(...),
                         db: Session = Depends(get_db)):
    data = db.query(CurrencyRate).filter(
        CurrencyRate.code == code.upper(),
        CurrencyRate.date >= start,
        CurrencyRate.date <= end,
        CurrencyRate.source == "NBU"
    ).order_by(CurrencyRate.date).all()
```

*Рисунок 3.4.6 – Отримання історії валютного курсу за діапазоном дат*

Усі маршрути захищено блоками try–except, а логування винесено у функцію log_message, що дозволяє зберігати помилки у файл журналу. У разі виникнення винятку система повертає HTTPException з відповідним повідомленням та кодом 500.

Загалом API реалізовано відповідно до REST-підходу, із підтримкою повного циклу CRUD-операцій для активів, збором і оновленням даних, а також параметризованим доступом до часових рядів. Така архітектура забезпечує ефективну взаємодію між усіма компонентами інструмента та відкриває можливості для подальшого розширення функціоналу.

### 3.5. Реалізація інтерфейсу

Інтерфейс користувача реалізовано за допомогою бібліотеки **Streamlit**, що дозволяє створювати сучасні вебзастосунки на Python без потреби в HTML/CSS/JS. Основна мета — надати користувачу простий і наочний доступ до фінансових даних у зручному форматі.

У межах інструменту інтерфейс охоплює увесь функціонал, реалізований на рівні API: перегляд валютних курсів, моніторинг фінансових активів, додавання нових об'єктів для відстеження, оновлення даних та побудова інтерактивних графіків.

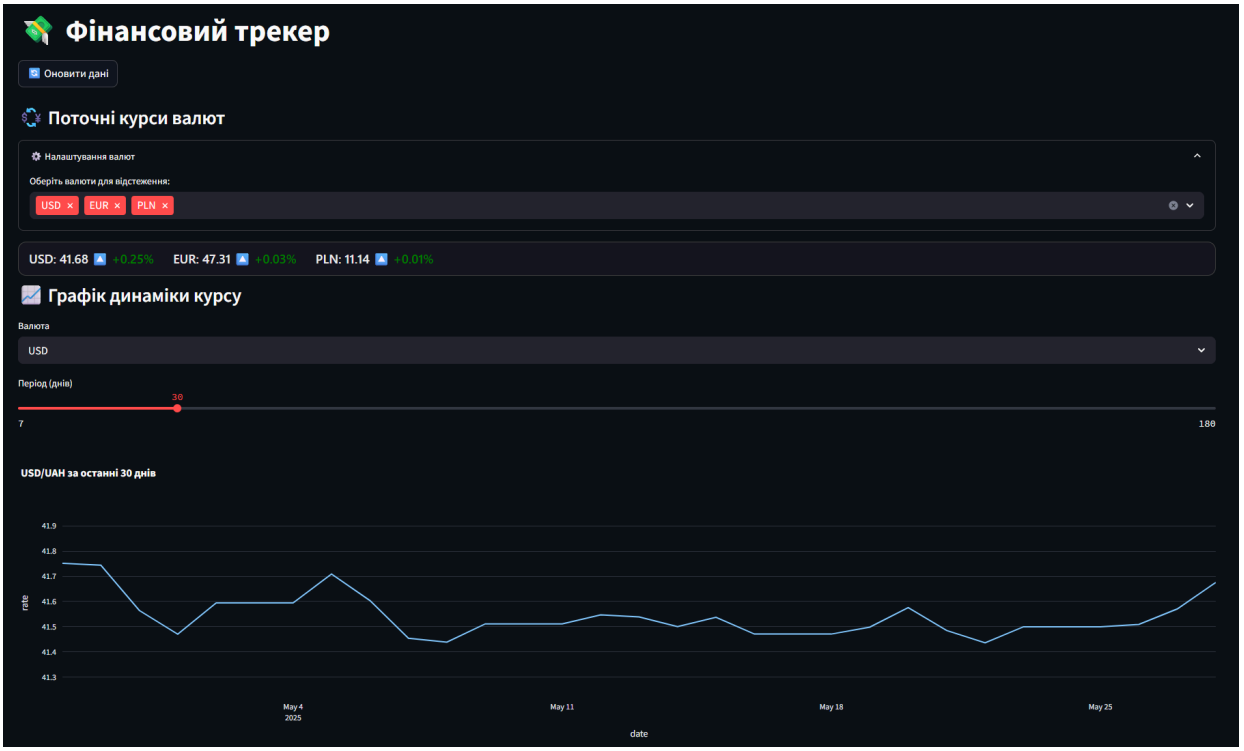


*Рисунок 3.5.1 – Головний інтерфейс із відображенням стрічки валют*

Одразу після запуску інструмент виконує **перевірку актуальності даних** за поточну дату. Якщо курси валют або котирування акцій відсутні, Streamlit надсилає запити до ендпоінтів /currency/update та /assets/update, автоматично ініціюючи збір нової інформації.

На головному екрані користувач бачить кнопку **«Оновити дані»**, яка виконує перезапуск сторінки з повторним завантаженням API. Нижче розташована **стрічка валют**, яка дозволяє обрати потрібні коди за допомогою multiselect. Усі курси виводяться у згорнутому вигляді: відображається поточне значення, відсоткова зміна за останню добу, кольорова індикація (зростання або спад), а також стрілка напрямку.

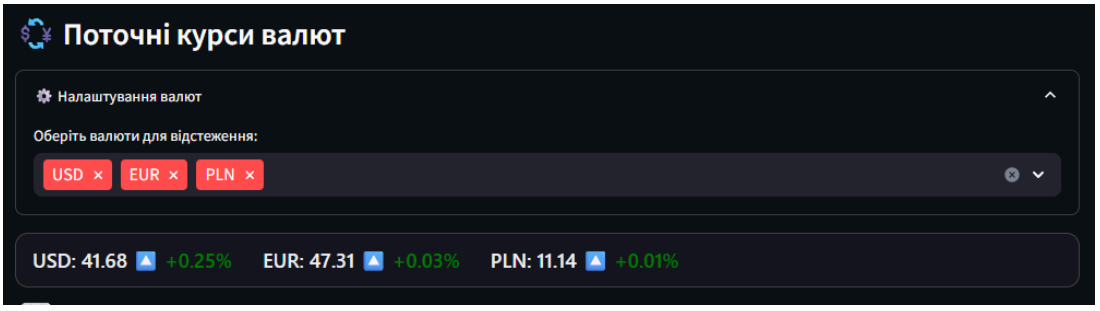

*Рисунок 3.5.2 – Стрічка курсів валют*

Далі доступна побудова **графіка динаміки валютного курсу**, який будується за допомогою бібліотеки plotly.express. Користувач обирає валюту, а також часовий період (від 7 до 180 днів), після чого відображається інтерактивний графік. Його можна масштабувати, гортати, наводити на точки для точних значень.

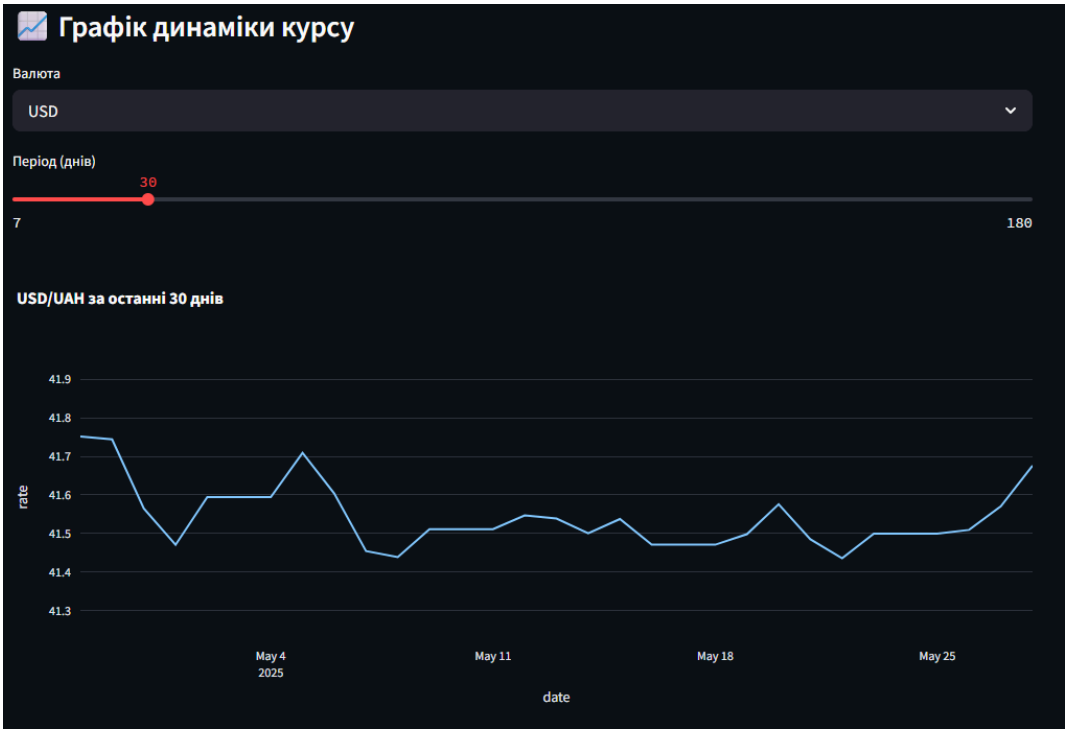


*Рисунок 3.5.3 – Графік валютного курсу*

На наступному екрані виводиться **стрічка поточних активів**. Вона реалізована як пагінована матриця елементів, де кожен актив відображає останню ціну та кнопку для побудови графіка. Перегортання реалізовано через form_submit_button, що дозволяє переходити по сторінках (по 6 активів за раз).

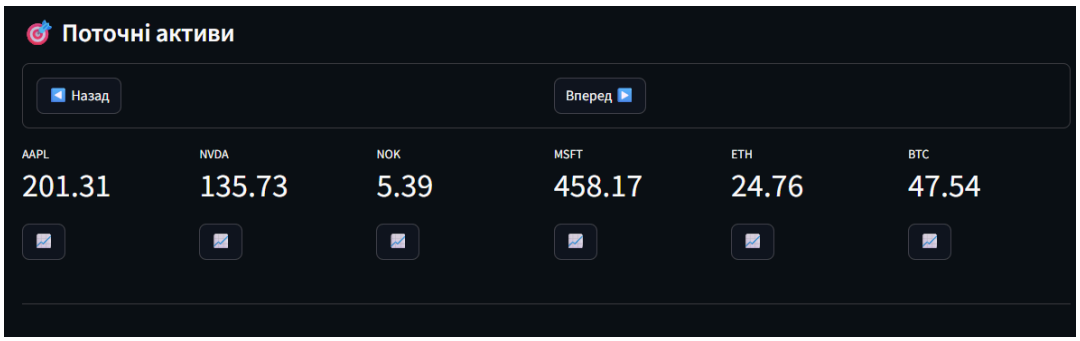


*Рисунок 3.5.4 – Стрічка активів із відображенням актуальних котирувань*

Після натискання на іконку 📈 активу виводиться **графік ціни активу**, центрований по ширині екрана. Користувач може обрати тип графіка — **Area** або **Candlestick**, що особливо зручно для технічного аналізу. Для реалізації графіків використано бібліотеки plotly.express та plotly.graph_objects.

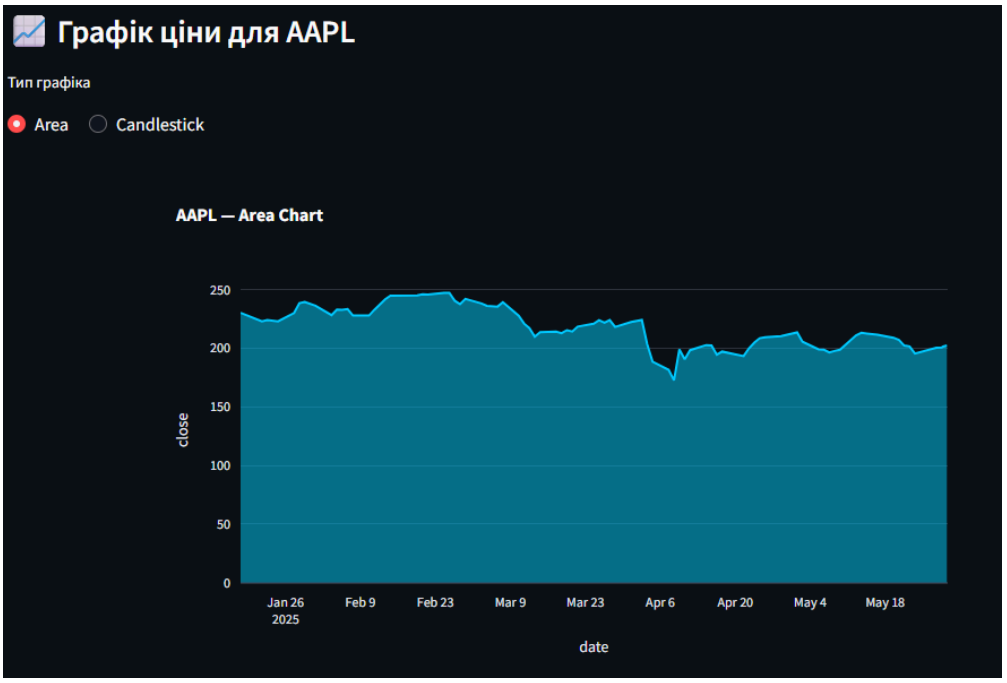

*Рисунок 3.5.5 – Графік ціни активу у форматі Area*


*Рисунок 3.5.6 – Графік ціни активу у форматі Candlestick*

Окремий блок праворуч дозволяє керувати активами. При введенні ≥3 символів здійснюється пошук через **Yahoo Finance API**, результати виводяться у вигляді селектора. Після вибору елементу можна додати актив до бази — для цього Streamlit надсилає POST-запит до /assets.

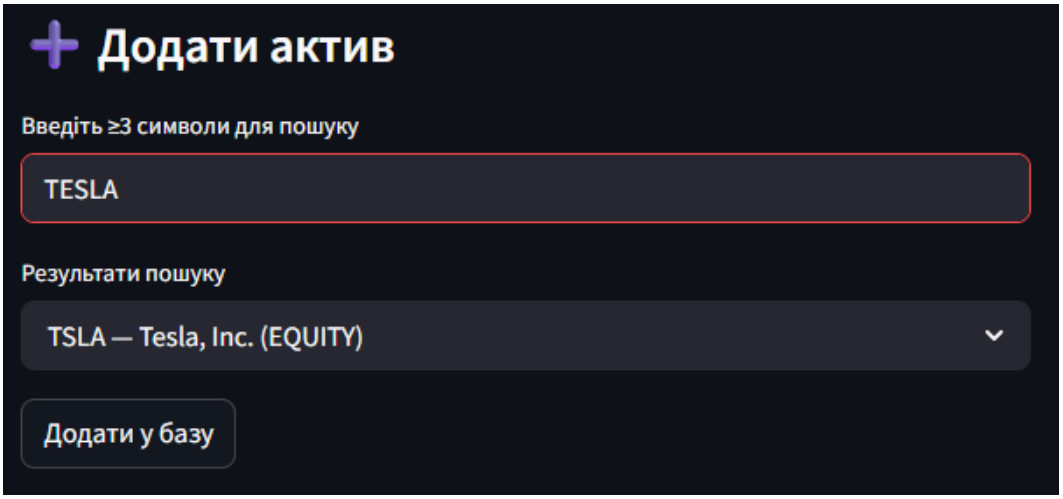


*Рисунок 3.5.7 – Пошук та додавання активу з Yahoo Finance*


*Рисунок 3.5.8 – Результати пошуку*

Для видалення передбачено ще один селектор, через який можна обрати актив зі списку, після чого здійснити запит на видалення (DELETE). Усі дії супроводжуються повідомленнями про успіх або помилку, що підвищує прозорість взаємодії.

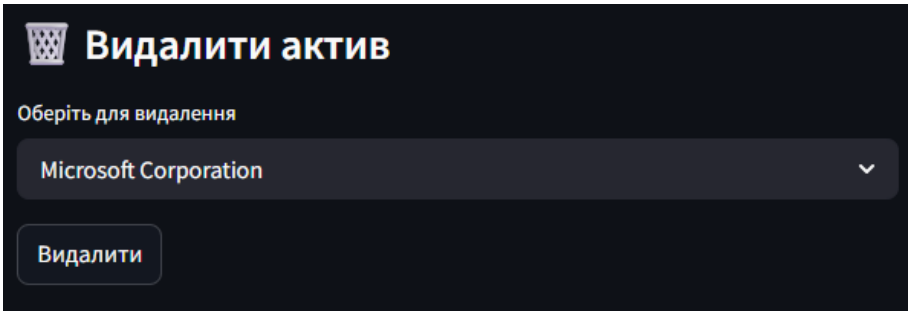


*Рисунок 3.5.9 – Видалення активу через інтерфейс Streamlit*

Уся взаємодія з інтерфейсом реалізована із застосуванням механізмів st.session_state, що дозволяє зберігати вибрані значення, поточну сторінку та активний графік. Це значно покращує UX при роботі з великими обсягами інформації.

Таким чином, реалізований інтерфейс забезпечує не лише повноцінну інтеграцію з API, а й зручний візуальний інструмент для щоденного використання. Він адаптивний, інтерактивний і повністю автоматизований — від збору до візуалізації даних.

# Висновки

У рамках виконання даної кваліфікаційної роботи було успішно розроблено та впроваджено автоматизований інструмент для інтеграції фінансових даних з різних джерел з метою їх подальшого аналізу та візуалізації.

Проведений у першому розділі огляд літератури дозволив системно дослідити історію розвитку проблеми інтеграції даних, визначити ключові підходи та технології, що застосовуються у цій галузі. Було встановлено, що незважаючи на численні досягнення, інтеграція даних залишається складною проблемою через гетерогенність джерел, питання забезпечення якості даних та підтримки їх актуальності.

У другому розділі було обґрунтовано вибір технологій та методів реалізації. Основою побудови рішення став інструмент, побудований за принципами багатошарової архітектури, з використанням класичного підходу ETL (Extract, Transform, Load), який забезпечив чітку послідовність етапів обробки даних. Для отримання інформації було обрано відкриті API фінансових сервісів (Yahoo Finance для акцій та криптовалют, НБУ для валютних курсів), для трансформації та збереження — стек Python-інструментів (requests, pandas, SQLAlchemy), з підключенням до бази даних Supabase, а для візуального представлення даних — фреймворк Streamlit.

У третьому розділі було детально проаналізовано реалізацію кожного з компонентів системи. Розроблено асинхронний механізм збору валютних даних, синхронну обробку котирувань активів, забезпечено перевірку правильності та унікальності даних при збереженні. Спроєктовано централізовану базу даних із унікальними обмеженнями та індексами, реалізовано REST API через FastAPI для взаємодії з даними, а також створено вебінтерфейс із підтримкою динамічного відображення,

автоматичного оновлення, фільтрації та побудови графіків у режимі реального часу.

Результатом стала єдина узгоджена система, яка дозволяє здійснювати повний цикл інтеграції даних — від збору до візуального аналізу — в інтерактивному середовищі. Користувач може переглядати динаміку валют, керувати списком активів, додавати нові елементи через інтеграцію з Yahoo, отримувати графіки в різних форматах (area та candlestick) та працювати з наочними візуалізаціями без потреби в технічних навичках.

У процесі роботи також було визначено перспективи розвитку системи. Зокрема, можлива автоматизація розширення списку валют і активів, підключення альтернативних джерел даних, оптимізація API через кешування, додавання аналітичних модулів та розширення персоналізації інтерфейсу.

Таким чином, мета дослідження — створення ефективного інструмента для інтеграції фінансових даних із подальшою можливістю аналітики та візуалізації — була досягнута повністю. Розроблений інструмент демонструє реальне застосування принципів інтеграції даних у практичній розробці та має значний потенціал для подальшого вдосконалення й використання в умовах реального середовища.

## Список використаних джерел


1. Bartley, K. Big data statistics: How much data is there in the world? [Електронний ресурс] / K. Bartley // Rivery. — 2024. — Режим доступу до ресурсу: https://rivery.io/blog/big-data-statistics-how-much-data-is-there-in-the-world/.
2. Spiceworks. What is Data Integration? Process, Tools, and Examples [Електронний ресурс]. – 2023. – Режим доступу: https://www.spiceworks.com/tech/devops/articles/data-integration/.
3. DigitalRoute. What Is Data Integration? [Електронний ресурс]. – Режим доступу: https://www.digitalroute.com/resources/glossary/data-integration/.
4. Deloitte. Data as a Strategic Asset [Електронний ресурс]. – 2021. – Режим доступу: https://www2.deloitte.com/us/en/pages/consulting/articles/data-strategic-asset.html.
5. TEKsystems. Unlocking the Hidden Value of Data as an Asset [Електронний ресурс]. – 2023. – Режим доступу: https://www.teksystems.com/en/insights/article/data-as-an-asset.
6. Psico-Smart. How can organizations leverage data analytics to enhance decision-making and improve performance? [Електронний ресурс]. – 2024. – Режим доступу: https://psico-smart.com/en/blogs/blog-how-can-organizations-leverage-data-analytics-to-enhance-de.
7. Haigh, T. How Data Got its Base: Information Storage Software in the 1950s and 1960s [Електронний ресурс]. – Режим доступу: https://www.tomandmaria.com/Tom/Writing/HowThe%20DataGotItsBase.pdf.

8. Ferranti Ltd. Ferranti Orion Computer System, 1960 [Електронний ресурс]. – Режим доступу: https://s3data.computerhistory.org/brochures/ferranti.orion.1960.102646248.pdf..
9. Cube.dev. The Evolution of OLAP [Електронний ресурс]. – 2024. – Режим доступу: https://cube.dev/blog/the-evolution-of-olap.
10. Inmon, W. H. Building the Data Warehouse. – Wiley, 1992.
11. Data Warehouse Definition – 1Keydata [Електронний ресурс]. – 2024. – Режим доступу: https://www.1keydata.com/datawarehousing/data-warehouse-definition.html.
12. GeeksforGeeks. ETL Tools Overview [Електронний ресурс]. – 2024. – Режим доступу: https://www.geeksforgeeks.org/etl-tools-overview/.
13. Waehner, K. The Past, Present and Future of Stream Processing [Електронний ресурс]. – 2024. – Режим доступу: https://www.kai-waehner.de/blog/2024/03/20/the-past-present-and-future-of-stream-processing/.
14. dbt Labs. Understanding ELT: Extract, Load, Transform [Електронний ресурс]. – 2024. – Режим доступу: https://www.getdbt.com/blog/extract-load-transform.
15. MuleSoft. iPaaS: Integration Platform as a Service Explained [Електронний ресурс]. – Режим доступу: https://www.mulesoft.com/integration/ipaas-integration-platform-as-a-service.
16. The CRO Club. 8 Integration Platform As A Service (iPaaS) Examples [Електронний ресурс]. – Режим доступу: https://croclub.com/data-reporting/ipaas-examples/.

17. Ideas2IT. MuleSoft Vs. Dell Boomi Vs. Zapier: Which is a better iPaaS? [Електронний ресурс]. – Режим доступу: https://www.ideas2it.com/blogs/mulesoft-vs-dell-boomi-vs-zapier-which-is-better.
18. New Relic. Serverless Architecture: Key Benefits and Limitations [Електронний ресурс]. – 2024. – Режим доступу: https://newrelic.com/blog/best-practices/what-is-serverless-architecture.
19. Estuary. 9 Best Python ETL Tools in 2025 [Електронний ресурс]. – 2025. – Режим доступу: https://estuary.dev/python-etl-tools/.
20. Apache Airflow. Apache Airflow® Documentation [Електронний ресурс]. – 2024. – Режим доступу: https://airflow.apache.org/.
21. Talend. Talend Data Integration — Software to Connect, Access, and Transform Data [Електронний ресурс]. – 2024. – Режим доступу: https://www.talend.com/products/integrate-data/.
22. В Машкіна, ВО Абрамов, ТІ Носенко, ЄВ Іваніченко.Навчально-методичний посібник до виконання і захисту кваліфікаційної роботи бакалавра спеціальностей 122 Комп’ютерні науки для здобувачів вищої освіти денної форми навчання, Київський столичний університет імені Бориса Грінченка. Київ, 2024.- 43 с.
23. Теоретичні та практичні аспекти використання математичних методів та інформаційних технологій в освіті й науці:моногр. / за заг. ред. О. Литвин. — К.: Київ. ун-т ім. Б. Грінченка,2021. — 332 с.
24. Замрій І. В., Шахматов І. О., Яскевич В. О. BlockchainSQLSecure: інтеграція блокчейн-технології для зміцнення захисту від SQL-ін'єкцій // Вісник Київського національного університету імені Тараса Шевченка. –2024. –No 1(29). –С. 160–167. –DOI: https://doi.org/10.17721/1812-5409.2024/1.29.
25. ОБ Придибайло, РВ Придибайло, Владислав Олександрович Яскевич, Юрій Владиславович Яскевич, «Архітектура нульової довіри: логічні компоненти та підходи запровадження», Зв’язок , № 3, 2024, стор 7-11

26. Airbyte. Airbyte | Open-Source Data Integration Platform | ELT Tool [Електронний ресурс]. – 2024. – Режим доступу: https://airbyte.com/.

27. Hevo. Hevo Features [Електронний ресурс]. – 2024. – Режим доступу: https://docs.hevodata.com/introduction/hevo-features/.

28. PostgreSQL Documentation. JSON Types [Електронний ресурс]. – 2024. – Режим доступу: https://www.postgresql.org/docs/current/datatype-json.html.

29. Google Cloud. Overview of BigQuery storage [Електронний ресурс]. – 2024. – Режим доступу: https://cloud.google.com/bigquery/docs/storage_overview..

30. RisingWave. Redshift vs Snowflake vs Google BigQuery: Comprehensive Comparison of Performance, Cost, and Usability [Електронний ресурс]. – 2024. – Режим доступу: https://risingwave.com/blog/redshift-vs-snowflake-vs-google-bigquery-comprehensive-comparison.

31. Supabase Docs. REST API [Електронний ресурс]. – 2024. – Режим доступу: https://supabase.com/docs/guides/api.

32. FastAPI. Features [Електронний ресурс]. – 2024. – Режим доступу: https://fastapi.tiangolo.com/features/.

33. Postman Blog. gRPC vs. GraphQL [Електронний ресурс]. – 2024. – Режим доступу: https://blog.postman.com/grpc-vs-graphql/.

34. A Systematic Review of Big Data Integration Challenges and Solutions // All Academic Research. – 2023. – Режим доступу: https://allacademicresearch.com/index.php/AJBAIS/article/view/111.

35. Challenges and Solutions in Data Integration for Heterogeneous Systems // ResearchGate. – 2024.

36. Data Quality Management in Big Data: Strategies, Tools, and Techniques // ScienceDirect. – 2025.

37. 11+ Most Common Data Integration Challenges & Solutions // Estuary. – 2025. – Режим доступу: https://estuary.dev/blog/data-integration-challenges/.
38. Top Trends for Data Streaming with Apache Kafka and Flink in 2025 // Medium. – 2025. – Режим доступу: https://kai-waehner.medium.com/top-trends-for-data-streaming-with-apache-kafka-and-flink-in-2025-636583892b2d.
39. Ontologies and Semantic Data Integration // ScienceDirect. – 2005.
40. Data Integration Challenges for Machine Learning in Precision Medicine // PMC. – 2022.
41. Semantic Data Integration and Querying: A Survey and Challenges // ACM Digital Library. – 2024.