ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___» _____ 202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
«СТВОРЕННЯ ІНТЕРАКТИВНОЇ ПЛАТФОРМИ ДЛЯ ВІЗУАЛІЗАЦІЇ МАСИВІВ ДАНИХ З ВИКОРИСТАННЯМ СУЧАСНИХ МЕТОДІВ МАШИННОГО НАВЧАННЯ»

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня магістра

Виконавець роботи Воробйов Ігор Олександрович
_____ «____»_____ 202_ р.
(підпис)

Науковий керівник к.пед.н., доцент, Кошова Оксана Петрівна
_____ «____»_____ 202_ р.
(підпис)

ПОЛТАВА 2025 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
(підпис)
«___» _____ 202__ р.

ЗАВДАННЯ І КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Створення інтерактивної платформи для візуалізації масивів даних з використанням сучасних методів машинного навчання»
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня магістр
Прізвище, ім'я, по батькові Воробйов Ігор Олександрович
Затверджена наказом ректора № ___-Н від «__»_____ 202_ р.
Термін подання студентом роботи «___»_____ 202__ р.
Вихідні дані до кваліфікаційної роботи: публікації по темі роботи, статті та документації з платформ для візуалізації масивів даних.

Зміст пояснювальної записки (перелік питань, які потрібно розробити)

ВСТУП

1. ПОСТАНОВКА ЗАДАЧІ
2. ОГЛЯД ПЛАТФОРМ АНАЛОГІЧНОГО ПРИЗНАЧЕННЯ
   2.1. Платформа для візуалізації даних Tableau.
   2.2. Платформа для ведення блогу Medium.
   2.3. Платформа для ведення блогу Ghost.
3. ТЕОРЕТИЧНА ЧАСТИНА
   3.1. Опис проектних рішень та інструментів розробки.
   3.2. Опис обраного підходу до створення програмного забезпечення.
4. ПРАКТИЧНА ЧАСТИНА
   4.1. Архітектура програмного забезпечення.
   4.2. Опис роботи платформи.
   4.3. Інструкція з використання платформи.

ВИСНОВКИ

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

ДОДАТОК А

Перелік графічного матеріалу: 38 ілюстрацій.

Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Кошова О.П.		
Інформаційний огляд	Кошова О.П.		
Теоретична частина	Кошова О.П.		
Практична частина	Кошова О.П.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка задачі		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доробка (за необхідністю), рецензування		

Дата видачі завдання «__» _____ 202___ р.

Здобувач вищої освіти _____ Воробйов Ігор Олександрович
(підпис)

Науковий керівник _____ к.пед.н., доц. Кошова О.П.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 202_ р.

Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 202__ р.

Погоджено
Науковий керівник _____
к.пед.н., Оксана КОШОВА
«____» _____ 202__ р.

План

кваліфікаційної роботи на тему
«Створення інтерактивної платформи для візуалізації масивів даних з використанням сучасних методів машинного навчання»
зі спеціальності 122 Комп'ютерні науки
освітня програма 122 «Комп'ютерні науки»
ступеня магістр
Прізвище, ім'я, по батькові Воробйов Ігор Олександрович

Здобувач вищої освіти _____ Ігор ВОРОБЙОВ
«____» _____ 202__ р.

РЕФЕРАТ

Записка: 123 сторінки, основна частина 80 сторінок, 38 рисунків, 1 додаток, 38 літературних джерел.

Мета роботи – створення інтерактивної платформи для візуалізації масивів даних з використанням сучасних методів машинного навчання.

Об’єкт розробки – інтерактивна платформа для візуалізації масивів даних з використанням сучасних методів машинного навчання.

Методи дослідження – використання технологій та інструментів розробки програмного забезпечення: інструмент проєктування інтерфейсів Figma, фреймворк React, бібліотека D3.js, фреймворк FastAPI, база даних PostgreSQL, фреймворк для моделювання PyTorch, редактор коду Visual Studio Code, розподілена система контролю версій Git.

Сформульовано вимоги до програмного забезпечення. Зроблено огляд платформ аналогічного призначення, виділені їх основні переваги та недоліки. Виконано опис інструментів та проектних рішень для розробки програмного забезпечення. Обрано методологію створення програмного забезпечення. Розроблено прототип інтерфейсу. Описано архітектуру програмного забезпечення. Побудовано діаграму прецедентів. Описано процедуру розгортання. Розроблено блог-платформу та інструкцію.

Ключові слова: FIGMA, REACT, D3JS, FASTAPI, PYTORCH, POSTGRESQL, ВІЗУАЛІЗАЦІЯ, МАШИННЕ НАВЧАННЯ.

ЗМІСТ

ВСТУП

Актуальність. Створення інтерактивної платформи для візуалізації масивів даних з використанням сучасних методів машинного навчання є актуальним через стрімке зростання обсягів даних у різних сферах діяльності, тому є вкрай велика потреба в інструментах, які перетворюють ці дані на зрозумілі, а також можуть спрогнозувати подальший розвиток. Все більше з'являється таких інструментів, що в свою чергу підкреслює попит на нові платформи та методи обробки масивів даних.

Візуалізація даних покращу якість прийняття рішень в різних сферах: бізнесі, медицині, державному управлінні, наукових дослідженнях. Це дозволяє швидко виявити аномалії, закономірності і тренди. Інтерактивні платформи знижують поріг входу в аналіз даних, роблячи це доступних не лише для фахових спеціалістів, а й для менеджерів та звичайних людей, що дозволяє оперативно приймати рішення. З огляну на зростання ринку BI та візуалізації, інвестиції в подібні рішення мають доволі високу цінність для організації різного масштабу.

Поєднання сучасних методів машинного навчання, а саме інтерактивне навчання, зниження розмірності, кластеризація, пояснювальні моделі, особливо з візуалізацією, відкриває нові можливості для інтерпретації складних структур даних. Наявність сучасного інструментарію для візуалізації також спрощує впровадження аналітики в операційну діяльність. Сьогодні існує багато інструментів та підходів, але потреба в кастомізованих інтерактивних рішеннях залишається високою.

Розробка такої платформи також має значення для підготовки фахівців з аналізу даних, вона може слугувати навчальною базою для практичних сценаріїв, інтерпретацією результатів та експериментів з алгоритмами.

Таким чином, актуальність теми обґрунтована поєднанням ринкових потреб, наукових трендів у машинному навчанні та практичної користі різних секторів життєдіяльності.

Метою роботи є створення інтерактивної платформи для візуалізації масивів даних з використанням сучасних методів машинного навчання.

Об'єктом розробки є платформа для візуалізації масивів даних з використанням сучасних методів машинного навчання.

Предметом розробки є програмне забезпечення, а саме інтерфейс програмування додатків, вебсайт для перегляду та керування даними.

Кваліфікаційна робота складається з чотирьох розділів, а саме: постановка задачі, огляд платформ аналогічного призначення, теоретичної та практичної частини. Структура побудована так, що дає змогу логічного представлення матеріалу та розкриття теми роботи.

Результатом виконання кваліфікаційної роботи є створення платформи для візуалізації масивів даних з використанням сучасних методів машинного навчання.

Обсяг пояснювальної записки: 123 сторінки, основна частина 80 сторінок, 38 рисунків, 1 додаток, 38 літературних джерела.

1. ПОСТАНОВКА ЗАДАЧІ

Необхідно розробити інтерактивну платформу для візуалізації масивів даних, яка поєднує зручний інтерфейс для користувача, масштабовану обробку великих наборів даних та модулі машинного навчання, для автоматичного виявлення закономірностей і побудови прогнозів. Платформа має забезпечувати інтерактивний аналіз, фільтрацію, зниження розмірності та пояснюваність результатів, щоб користувачі різного рівня підготовки могли швидко отримувати інформацію з даних.

Розроблена інтерактивна платформа для візуалізації масивів даних повинна складатися з наступних сторінок:

- авторизація;
- головна панель;
- список наборів;
- детальний набір даних;
- побудова візуалізації.

Сторінка «Авторизація». Сторінка для безпечного входу користувачів і відновлення доступу. Містить форму входу, поле для відновлення пароля та механізм редіректу на головну панель після автентифікації. Забезпечує обробку токенів (JWT) і базову валідацію введених даних.

Сторінка «Головна панель». Стартова сторінка з оглядом системи і швидким доступом до основних дій. Показує список наборів даних і кнопки швидких дій (завантажити дані, новий аналіз). Служить точкою входу до подальшої роботи з платформою.

Сторінка «Список наборів». Сторінка для керування наборами даних: перегляд, завантаження та базова інформація. Містить таблицю наборів з метаданими, прев'ю перших рядків і статус обробки. Дозволяє імпортувати CSV/JSON і видаляти або версіонувати набори.

Сторінка «Детальний набір даних». Детальний огляд конкретного набору даних і інструменти попередньої обробки. Включає табличне прев'ю з фільтрами, базову статистику по стовпцях і прості операції очищення (заповнення пропусків, нормалізація). Має кнопку для запуску аналізу або експорту підмножин.

Сторінка «Побудова візуалізації». Головна інтерактивна сторінка для візуалізації та запуску ML-задач. Показує інтерактивний графік (scatter) з можливістю вибору підмножин, панель вибору методів (PCA, UMAP, KMeans, IsolationForest) і відображення результатів (кластерні мітки, аномалії). Дозволяє застосувати результати до набору даних і експортувати метадані.

2. ОГЛЯД ПЛАТФОРМ АНАЛОГІЧНОГО ПРИЗНАЧЕННЯ

На даний момент у світі існує значне число платформ візуалізації масивів даних, тому далі ми розглянемо деякі з них.

2.1. Платформа для роботи з даними Tableau

Tableau – це комплексна платформа для візуалізації та дослідження даних, призначена для швидкого створення інтерактивних дашбордів, ad-hoc аналізу та поширення аналітики в організації; платформа підтримує як локальні (on-prem) розгортання, так і хмарні інсталяції, дозволяючи запускати Tableau Server у приватних хмарах або використовувати Tableau Cloud для хостингу сервісів (Рисунок 2.1).

Рисунок 2.1 – Платформа для роботи з даними «Tableau»

Tableau має багаторівневу архітектуру: клієнтська частина (Desktop/Web), шар обробки/серверних процесів і шар зберігання/джерел даних. При підключенні Tableau може виконувати обчислення локально (extracts) або делегувати їх безпосередньо джерелу даних (live connections), що дає гнучкість між продуктивністю та актуальністю даних. Серверні процеси (Gateway, VizQL, Data Engine, Backgrounder тощо) координують рендеринг візуалізацій, кешування, виконання задач підготовки даних і планування задач [1].

Ключові можливості аналітики та візуалізації:

− візуалізації. Широкий набір типів (карти, лінійні/стовпчикові графіки, heatmap, treemap, boxplot тощо) з інтуїтивним drag-and-drop інтерфейсом для швидкого прототипування;

− підготовка даних. Інтегрований Tableau Prep для ETL-операцій, можливість створювати extracts для пришвидшення запитів або використовувати live-з'єднання для актуальних даних;

− аналітичні інструменти. Обчислювані поля, параметри, вбудовані статистичні функції, прогнозування (forecasting), кластеризація, можливості для кастомних розрахунків і сценаріїв «what-if»;

− Self-service і storytelling. Можливість створювати інтерактивні історії (Story), ділитися шаблонами і надавати бізнес-користувачам інструменти self-service аналітики;

− AI/помічники. Інструменти для автоматичного підбору візуалізацій, Explain Data та інші функції, що допомагають інтерпретувати аномалії й тренди (залежно від версії і ліцензії).

Tableau пропонує понад 10 окремих продуктів, і їхня кількість продовжує зростати. Ці продукти можна поділити на дві основні категорії. Перша категорія включає продукти, зосереджені на розробці, зокрема Tableau Server, тоді як друга категорія охоплює продукти, призначені для публікації, зокрема Tableau Desktop. Деякі з цих продуктів мають подібні інтерфейси та перетин функціональності, проте кожен має власну архітектуру. Хоча всі вони інтегровані в одну екосистему, деякі створені для тіснішої взаємодії між собою. Tableau Desktop і Tableau Server – яскравий приклад: те, що створюється в Tableau Desktop, передається до Tableau Server, де може бути опубліковане, збережене, кероване та доступне для спільної роботи.

Архітектура Tableau Desktop є відносно простою. Вона складається з трьох основних шарів: шар даних (Data Layer), шар обчислень (Calculations Layer), шар візуалізацій (Visualizations Layer).

Шар даних – це місце, куди надходять дані в Tableau Desktop і де вони готуються до аналізу та візуалізації. Те, як він працює, залежить від того, як під'єднуються до даних, адже існує два типи підключень:

− Extract Connection (екстракт-підключення). Створює статичний знімок даних, який зберігається локально всередині проєкту;

− Live Connection (живе підключення). Напряму виконує запити до оригінального джерела даних у режимі реального часу щоразу, коли йде взаємодія з будь-якою візуалізацією.

Незалежно від типу підключення, Tableau Desktop пропонує 90 конекторів даних, щоб отримувати доступ до даних, де б вони не зберігалися – у плоских файлах, SQL-серверах чи хмарних платформах керування документами. Шар даних також зберігає інформацію про доступ до даних, таку як розташування та облікові дані авторизації, і забезпечує можливості моделювання даних, наприклад агрегування та об'єднання таблиць або зміну типів даних.

Tableau має потужний обчислювальний механізм, який дозволяє створювати користувацькі поля для виконання аналізу, необхідного для ваших візуалізацій – від агрегування, сегментації, фільтрації, перетворення типів даних до інших операцій.

Він пропонує три основні типи обчислень:

− Basic Calculations (базові обчислення). Дозволяють маніпулювати даними на рівні рядка або на агрегованому рівні;

− Level of Detail Expressions (LOD-вирази). Дають змогу виконувати обчислення з контролем рівня деталізації;

− Table Calculations (табличні обчислення). Дозволяють виконувати агрегування по всій таблиці або по певному набору рядків.

Шар візуалізацій – це місце, де можна створювати, налаштовувати та форматувати візуалізації. Цей шар пропонує три ієрархічні структури для створення візуальних елементів:

− Worksheet (робочий аркуш). Базовий будівельний блок у Tableau. Це полотно, де ви можете створювати, фільтрувати та форматувати будь-які діаграми;
− Dashboard (інформаційна панель). Полотно, де можна поєднувати один або кілька робочих аркушів і визначати їхню взаємодію, а також додавати інші об’єкти - текст, зображення, кнопки, вбудовані вебсторінки тощо;
− Story (історія). Місце, де можна об’єднати кілька панелей, щоб користувачі могли переглядати їх як один зв’язний звіт.

Три шари Tableau Desktop – дані, обчислення та візуалізації – разом утворюють робочу книгу Tableau (workbook). Її можна локально зберегти всю робочу книгу як файл Tableau packaged workbook (.twbx) і поділитися ним з іншими членами команди для перегляду та співпраці [2].

Приклад використання платформи на основі моніторингу якості повітря (Рисунок 2.2), на основі бази даних з продажів (Рисунок 2.3), статистика роботи відділу кадрів (Рисунок 2.4).

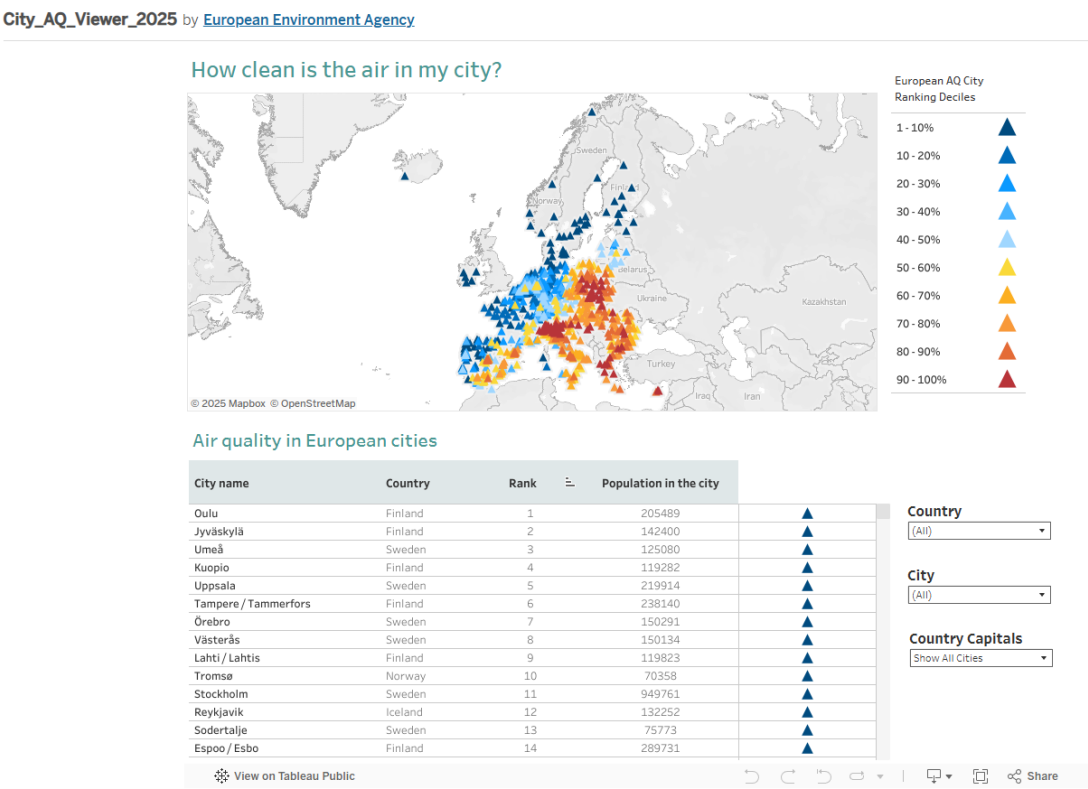


Рисунок 2.2 – Графік якості повітря на платформі «Tableau»

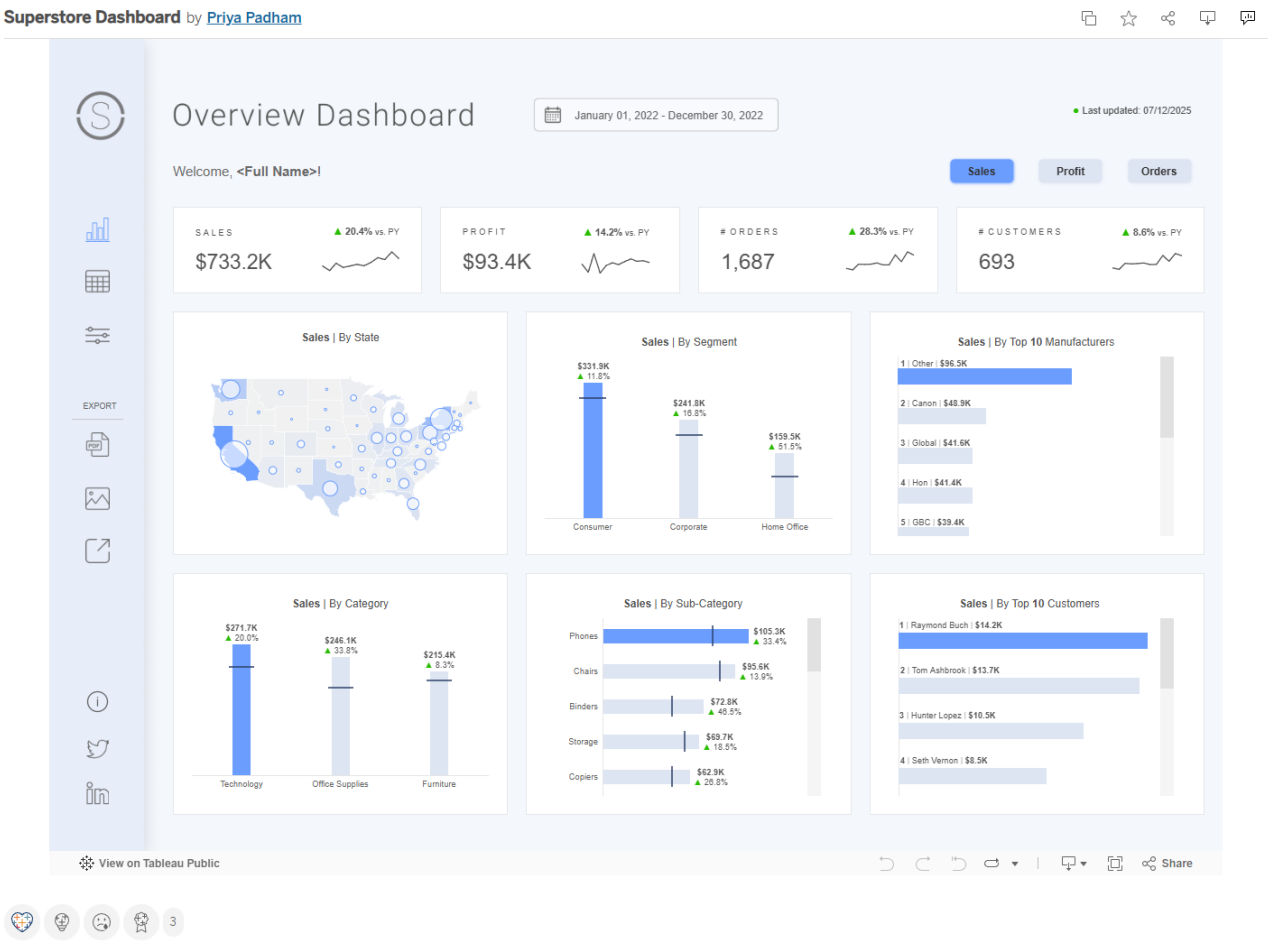


Рисунок 2.3 – Статистика продажів на платформі «Tableau»

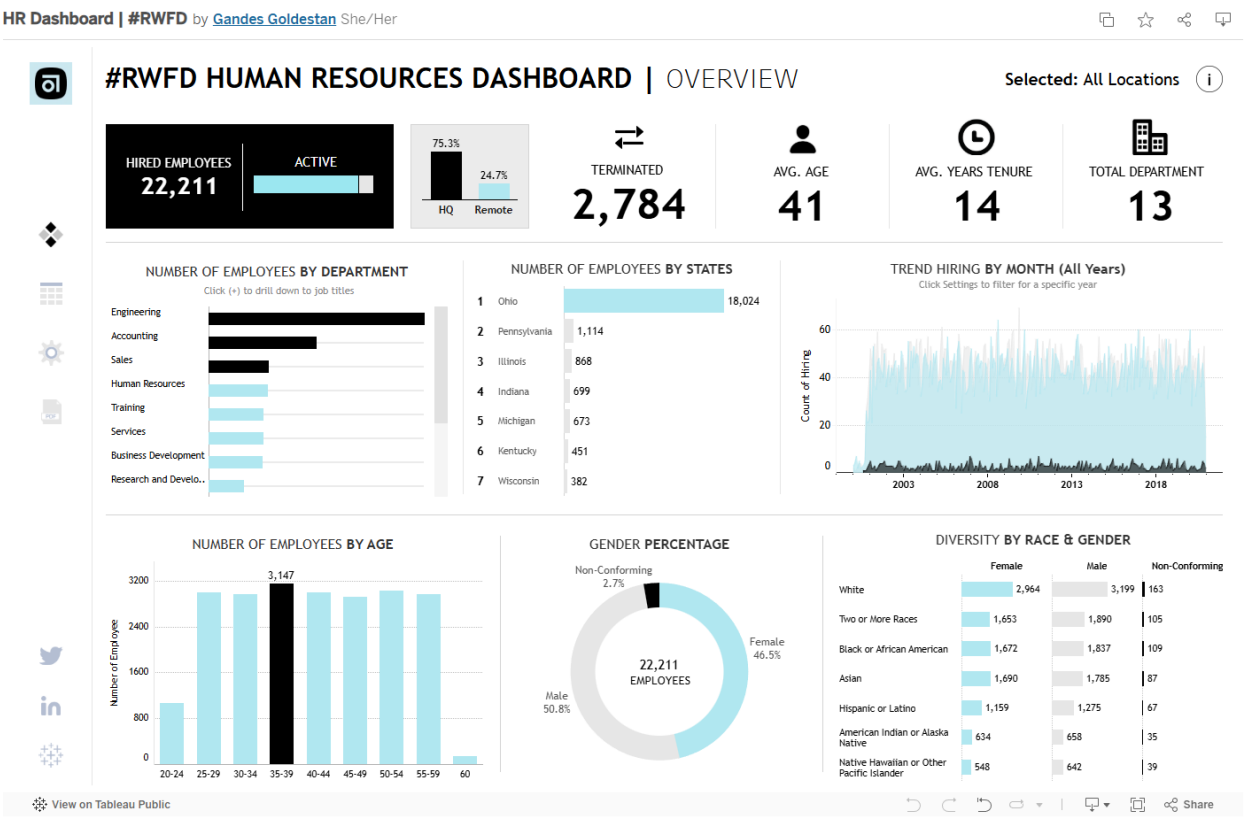


Рисунок 2.4 – Статистика відділу кадрів на платформі «Tableau»

Переваги користування [3]:

– Конструктор дашбордів Drag-and-Drop. Багато користувачів кажуть, що Tableau дозволяє легко створювати дашборди без написання коду. Інтерфейс виглядає візуальним, гнучким та інтуїтивним, особливо коли ви звикнете до нього.

– Швидке підключення до даних та їх дослідження. Tableau швидко підключається майже до будь-якого джерела даних — від електронних таблиць і баз даних до хмарних застосунків. Це означає, що ви можете перейти від сирих даних до аналізу за лічені хвилини.

– Красиві візуалізації та сторітелінг. Платформа відома тим, що створює чисті, інтерактивні графіки, які полегшують розуміння даних не лише аналітикам, а й клієнтам та стейкхолдерам.

− Легко працює з великими наборами даних. Кілька користувачів зазначили, що використовують Tableau для роботи з великими обсягами даних, і воно працює добре навіть із мільйонами рядків.

− Вбудована адаптивність для мобільних пристроїв. Дашборди автоматично підлаштовуються під різні екрани, що корисно, якщо команда переглядає звіти з телефонів або планшетів.

Недоліки користування:

− Крута крива навчання. Якщо ви новачок у BI-інструментах, Tableau може здатися складним. Багато користувачів кажуть, що потрібно часом навіть формальне навчання, щоб повністю зрозуміти такі функції, як обчислювані поля, LOD-вирази та розширені фільтри.

− Ціна може бути бар'єром. Одне з найпоширеніших зауважень — вартість. Для невеликих команд або стартапів Tableau може бути дорогим, особливо якщо потрібно додати багато користувачів, хостинг сервера або розширені функції.

− Не дуже дружній для початківців із простими задачами. Якщо вам потрібно просто відстежувати кілька ключових метрик або створити простий дашборд, Tableau може здатися надмірним. Деякі користувачі кажуть, що інструмент більше підходить для досвідчених аналітиків або великих команд.

− Додаткова залежність від сторонніх інструментів. Деякі користувачі зазначають, що їм потрібні інші інструменти для ETL, контролю версій або глибшої співпраці, що збільшує витрати на налаштування та підтримку.

− Повільніша робота при складній кастомізації. Хоча Tableau добре працює з більшістю даних, користувачі повідомляють про затримки або уповільнення, коли дашборди стають надто складними. Забагато фільтрів, візуальних шарів або змішаних джерел може вплинути на продуктивність.

2.2. Платформа для роботи з даними Looker

Looker – це сучасна аналітична платформа, побудована навколо ідеї семантичного шару, який дозволяє організаціям централізовано визначати бізнес-метрики, логіку обчислень і зв’язки між даними (Рисунок 2.5).

Рисунок 2.5 – Платформа для роботи з даними «Looker»

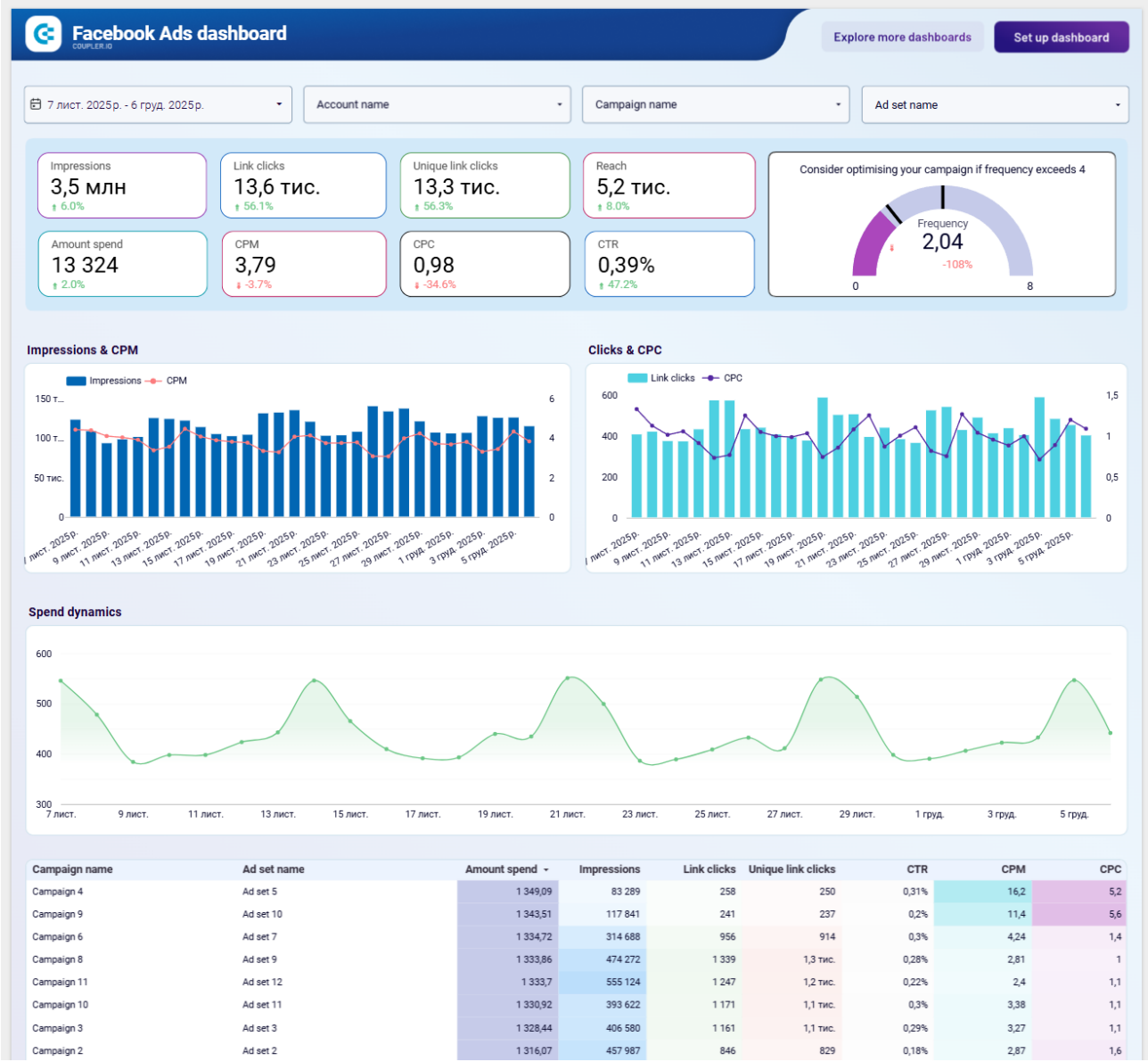
На відміну від традиційних BI-інструментів, Looker не копіює дані у власне сховище, а працює безпосередньо з аналітичними базами даних і хмарними data warehouse, виконуючи SQL-запити в реальному часі. Такий підхід забезпечує узгодженість метрик, відсутність дублювання даних і масштабованість, що залежить від потужності сховища, а не від самої платформи. Основою Looker є LookML – модельна мова, яка дозволяє описувати структуру даних, визначати вимірювання, агрегації, правила доступу та бізнес-логіку. LookML працює як шар абстракції над SQL, забезпечуючи повторне використання моделей, контроль версій через Git і можливість командної розробки. Це створює єдине джерело правди для всієї організації, де всі звіти та дашборди базуються на узгоджених метриках [4].

Користувачі взаємодіють із даними через інтерфейс Explore, який дозволяє будувати запити без написання SQL, комбінувати поля з різних таблиць, створювати візуалізації та зберігати їх як Looks або дашборди. Дашборди в Looker підтримують інтерактивні фільтри, drill-down, кастомні візуалізації та можливість вбудовування у сторонні веб-додатки. Платформа

також має API та SDK, що дозволяють автоматизувати робочі процеси, створювати власні аналітичні інструменти та інтегрувати Looker у внутрішні системи компанії. Завдяки Looker Extensions Framework можна створювати повноцінні додатки поверх даних, використовуючи Looker як бекенд для аналітики.

Looker активно використовується для централізованої аналітики, операційного моніторингу, вбудованої аналітики та створення кастомних data-додатків. Його сильна сторона – можливість забезпечити узгодженість метрик у великих організаціях, де різні команди працюють із одними й тими ж даними. Платформа підтримує гнучкі механізми безпеки, включаючи контроль доступу на рівні рядків, інтеграцію з SSO та детальне управління ролями. Водночас Looker вимагає технічних навичок для створення LookML-моделей, а його вартість орієнтована на корпоративний сегмент [5].

Приклад використання платформи на основі рекламної статистики в соціальній мережі Facebook (Рисунок 2.6), на основі статистики активності користувачів з Google Analytics (Рисунок 2.7), маркетингова статистика (Рисунок 2.8).



Campaign name	Ad set name	Amount spend	Impressions	Link clicks	Unique link clicks	CTR	CPM	CPC
Campaign 4	Ad set 5	1 349,09	83 289	258	250	0,31%	16,2	5,2
Campaign 9	Ad set 10	1 343,51	117 841	241	237	0,2%	11,4	5,6
Campaign 6	Ad set 7	1 334,72	314 688	956	914	0,3%	4,24	1,4
Campaign 8	Ad set 9	1 333,86	474 272	1 339	1,3 тис.	0,28%	2,81	1
Campaign 11	Ad set 12	1 333,7	555 124	1 247	1,2 тис.	0,22%	2,4	1,1
Campaign 10	Ad set 11	1 330,92	393 622	1 171	1,1 тис.	0,3%	3,38	1,1
Campaign 3	Ad set 3	1 328,44	406 580	1 161	1,1 тис.	0,29%	3,27	1,1
Campaign 2	Ad set 2	1 316,07	457 987	846	829	0,18%	2,87	1,6

Рисунок 2.6 – Рекламна статистика на платформі «Looker»

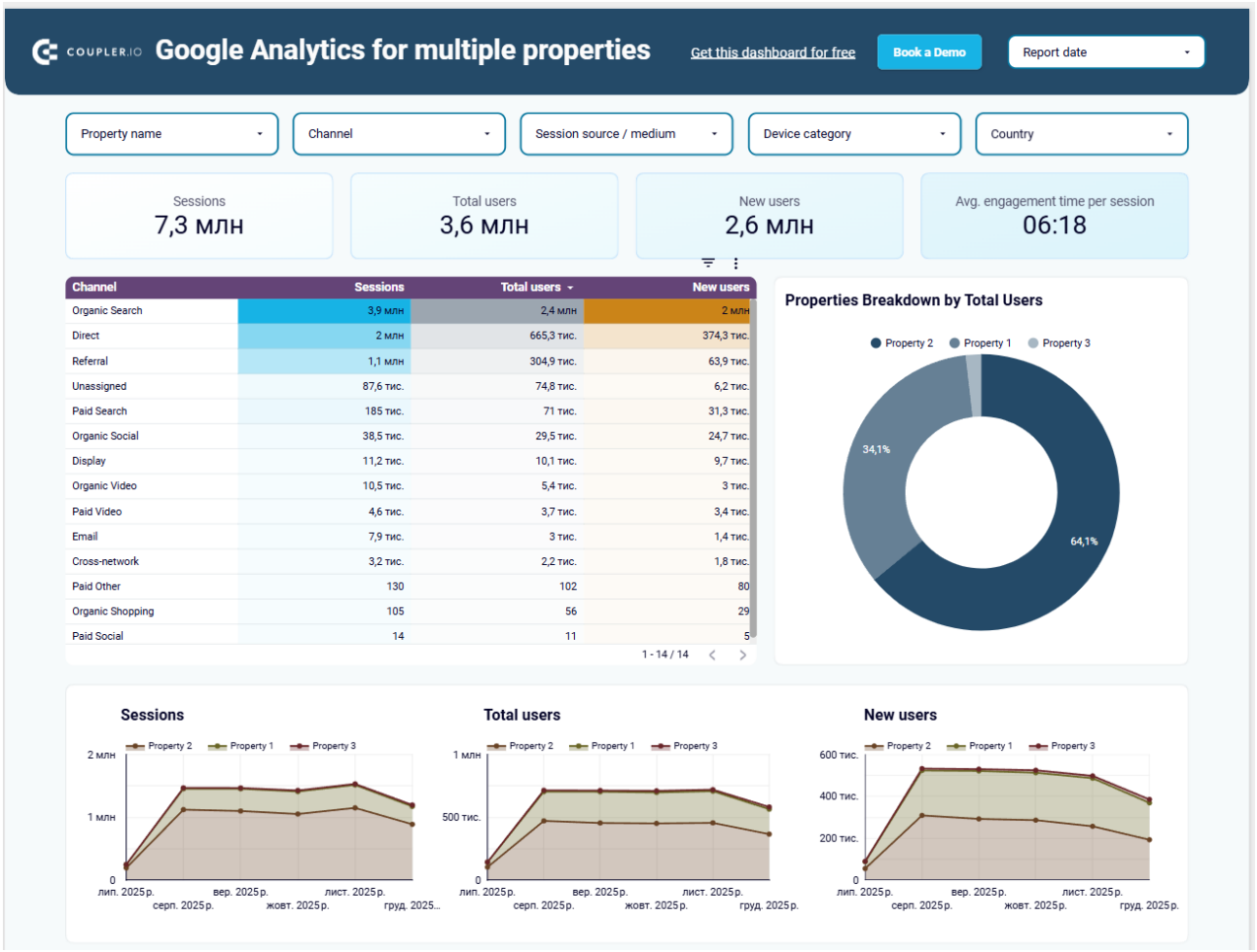


Рисунок 2.7 – Статистика активності користувачів з Google Analytics на платформі «Looker»

Рисунок 2.8 – Маркетингова статистика на платформі «Looker»

Переваги користування [6]:

– Єдиний семантичний шар (LookML). LookML дозволяє централізовано визначати бізнес-метрики, правила агрегації та зв'язки між таблицями. Це усуває хаос у звітах і забезпечує «єдину правду» для всієї організації.

– SQL-перший підхід без дублювання даних. Looker не зберігає дані у власному сховищі – усі обчислення виконуються безпосередньо в data warehouse. Це означає актуальні дані, відсутність extracts і масштабованість, що залежить від DW.

− Глибока інтеграція з хмарними сховищами. Платформа ідеально працює з BigQuery, Snowflake, Redshift та іншими сучасними аналітичними базами, використовуючи їхню продуктивність.

− Гнучкі можливості вбудовування (embedded analytics). Looker легко інтегрується у веб-додатки, дозволяє створювати кастомні аналітичні продукти та data-додатки поверх своїх API.

− API та SDK. Платформа підтримує автоматизацію, кастомні візуалізації, інтеграцію з CI/CD, створення власних інструментів і розширень.

− Контроль доступу та безпека. Looker підтримує SSO, контроль доступу на рівні рядків, рольову модель і Git-версіонування моделей.

− Explore-інтерфейс для аналітиків. Аналітики можуть будувати запити без SQL, комбінувати поля, створювати візуалізації та дашборди без участі інженерів.

Недоліки користування:

− Потребує технічних навичок для LookML. Створення моделей вимагає знань SQL і розуміння структури даних. Це не «drag-and-drop» інструмент, як Tableau чи Power BI.

− Висока вартість. Looker орієнтований на enterprise-сегмент. Ліцензування дороге, особливо для великих команд.

− Залежність від продуктивності data warehouse. Якщо DW повільний або неправильно налаштований, Looker також працюватиме повільно, бо всі запити виконуються в базі.

− Менше візуалізацій «із коробки». Looker має базовий набір графіків. Для складних або дизайнерських візуалізацій потрібні кастомні компоненти або сторонні бібліотеки.

− Вища складність впровадження. Потрібно налаштувати LookML-моделі, структуру проектів, Git-процеси, правила доступу – це займає час і ресурси.

− Не підходить для швидких прототипів. Через необхідність моделювання даних Looker повільніший у старті, ніж інструменти типу Metabase чи Superset.

2.3. Платформа для роботи з даними Grafana

Grafana – це потужна платформа для візуалізації даних, моніторингу та аналітики, яка стала де-факто стандартом у сфері observability. Її головна ідея полягає в тому, що вона не зберігає дані у власному сховищі, а підключається до зовнішніх джерел – таких як Prometheus, Loki, Elasticsearch, InfluxDB, PostgreSQL, MySQL, Tempo та десятки інших. Завдяки цьому Grafana працює як універсальний інтерфейс для перегляду метрик, логів і трасувань, об’єднуючи їх у єдиному просторі. Це дозволяє командам DevOps, SRE, інженерам підтримки та аналітикам отримувати повну картину стану систем у реальному часі (Рисунок 2.9).

Рисунок 2.9 – Платформа для роботи з даними «Grafana»

Архітектура Grafana побудована навколо концепції data sources — підключених систем, з яких Grafana отримує дані через API або SQL-запити. Кожне джерело може мати власний тип даних: часові ряди, логи, трасування, бізнес-метрики або навіть звичайні таблиці. Grafana не виконує важких обчислень – вона делегує їх джерелу даних, а сама відповідає за рендеринг графіків, побудову дашбордів і взаємодію з користувачем. Такий підхід робить платформу надзвичайно масштабованою: продуктивність залежить від джерела даних, а не від самої Grafana.

Однією з ключових особливостей Grafana є її візуалізаційні можливості. Платформа підтримує десятки типів графіків – від класичних лінійних і стовпчикових до heatmap, gauge, geomaps, node-graphs і складних панелей для APM. Кожна панель має гнучкі налаштування, включаючи трансформації

даних, порогові значення, анотації, комбінування серій і кастомні запити. Дашборди можуть бути інтерактивними: користувачі застосовують фільтри, перемикають часові діапазони, додають змінні та створюють шаблони для повторного використання. Це робить Grafana зручним інструментом як для оперативного моніторингу, так і для глибокого аналізу [7].

Grafana також є центральним елементом повного стеку observability, який включає Prometheus для метрик, Loki для логів і Tempo для трасувань. У такій конфігурації платформа дозволяє корелювати події: наприклад, користувач може перейти від піку CPU до відповідних логів або трасувань запитів.

Grafana дозволяє створювати правила сповіщень, які можуть базуватися на метриках, логах або комбінованих умовах. Сповіщення можуть надсилатися в Slack, Teams, PagerDuty, email або будь-яку іншу систему через webhook. Нова система Alerting (Grafana Unified Alerting) об’єднує правила з різних джерел у єдиному інтерфейсі, що спрощує управління складними інфраструктурами.

Платформа має розширювану архітектуру: користувачі можуть встановлювати плагіни для нових джерел даних, панелей або інтеграцій. Це дозволяє адаптувати Grafana під будь-які потреби – від IoT-моніторингу до бізнес-аналітики. Крім того, Grafana підтримує provisioning – автоматичне створення дашбордів, джерел даних і користувачів через YAML-конфігурації, що робить її зручною для DevOps-підходів і CI/CD.

У сфері безпеки Grafana підтримує SSO, OAuth, LDAP, granular permissions і розмежування доступу до папок, дашбордів і джерел даних. Це дозволяє використовувати її в корпоративних середовищах з високими вимогами до контролю доступу. Платформа доступна як у вигляді open-source версії, так і в комерційних редакціях Grafana Enterprise та Grafana Cloud, які додають функції аудиту, розширеного керування користувачами, масштабування та підтримки [8].

Приклад використання платформи на основі статистики з продажів (Рисунок 2.10), на основі активності в платформі Jira (Рисунок 2.11), статистика витрат (Рисунок 2.12).

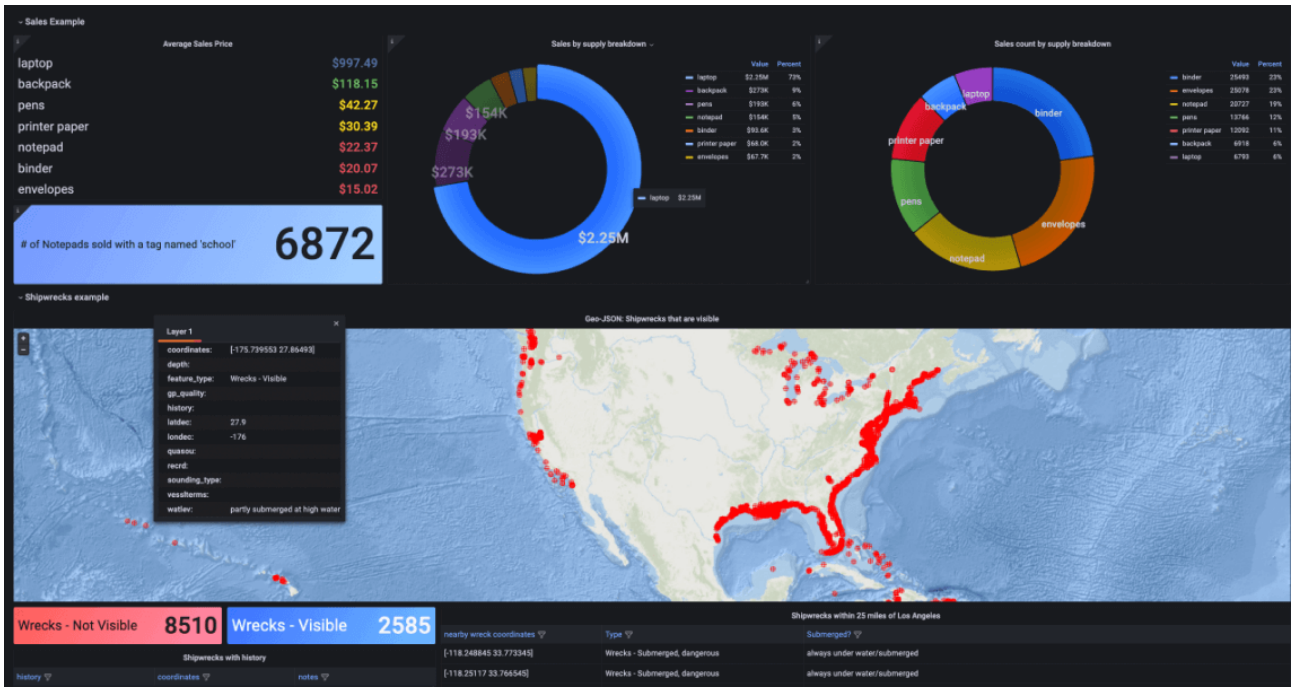


Рисунок 2.10 – Статистики з продажів на платформі «Grafana»

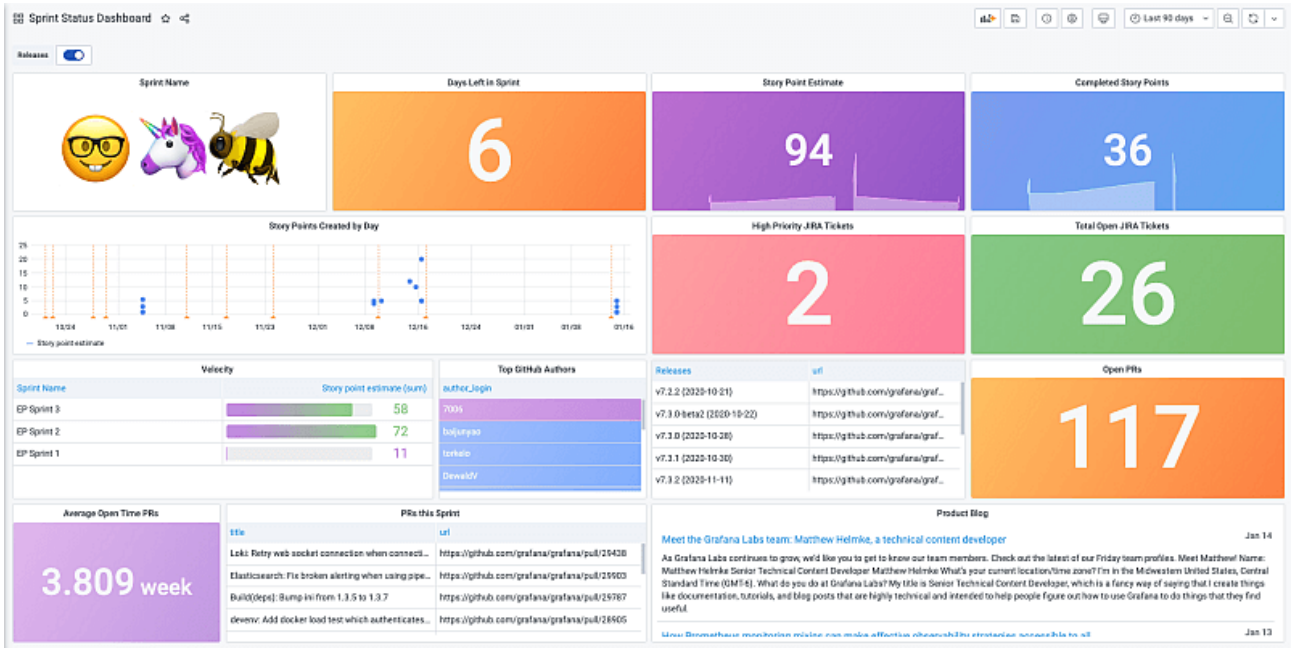


Рисунок 2.11 – Статистика з платформи «Jira» на платформі «Grafana»

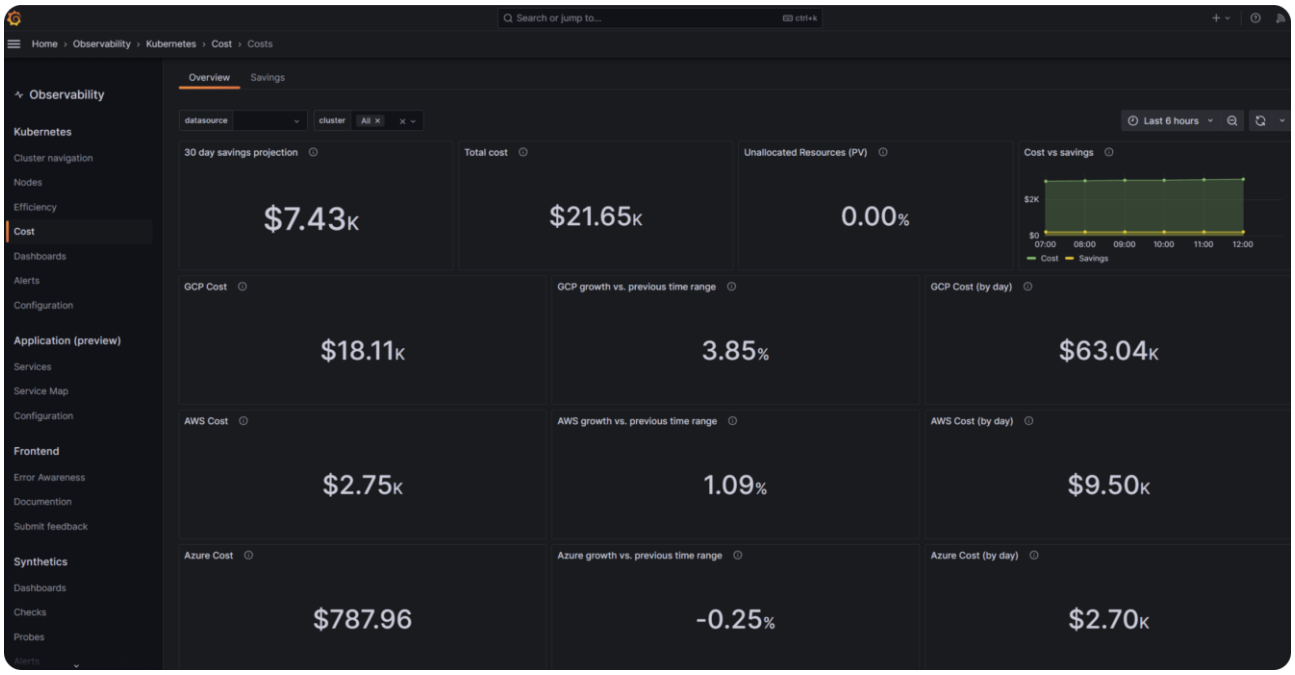


Рисунок 2.12 – Статистика витрат на платформі «Grafana»

Переваги користування [9]:

– Підтримка великої кількості джерел даних. Grafana може підключатися до Prometheus, Loki, Elasticsearch, InfluxDB, SQL-баз, хмарних сервісів та десятків інших систем, що робить її універсальним інструментом для будь-якої інфраструктури.

– Потужні можливості візуалізації. Платформа пропонує широкий набір графіків і панелей, гнучкі налаштування, змінні, фільтри та анотації, що дозволяє створювати складні та інформативні дашборди.

– Центральний елемент стеку observability. У поєднанні з Prometheus, Loki та Tempo Grafana забезпечує повний цикл спостереження: метрики, логи та трасування в одному інтерфейсі.

– Гнучка система алертингу. Grafana дозволяє створювати складні правила сповіщень і надсилати їх у Slack, Teams, PagerDuty, email або будь-яку іншу систему через webhook.

– Розширюваність через плагіни. Платформа підтримує плагіни для нових джерел даних, панелей і інтеграцій, що дозволяє адаптувати її під специфічні потреби.

− Підтримка автоматизації та DevOps-підходів. Provisioning через YAML-конфігурації дозволяє автоматично створювати дашборди, джерела даних і користувачів у CI/CD-процесах.

Недоліки користування:

− Залежність від продуктивності джерел даних. Оскільки Grafana не зберігає дані, швидкість роботи повністю залежить від того, наскільки швидко відповідає база або система моніторингу.

− Високий поріг входу для новачків. Налаштування джерел даних, написання запитів, робота з PromQL або Elasticsearch-квері вимагають технічних знань.

− Обмежені можливості для бізнес-аналітики. Grafana чудово підходить для моніторингу, але не для складних бізнес-звітів, фінансових моделей або багатовимірної аналітики.

− Складність адміністрування у великих організаціях. Налаштування ролей, доступів, SSO та інтеграцій може вимагати значних зусиль від DevOps-команди.

− Частина функцій доступна лише у комерційних версіях. Розширений аудит, корпоративні інтеграції та масштабування доступні тільки в Grafana Enterprise або Grafana Cloud.

2.4. Платформа для роботи з даними Kibana

Kibana – це аналітична та візуалізаційна платформа, створена як інтерфейс для роботи з Elasticsearch і тісно інтегрована в екосистему Elastic Stack. Її основне призначення – надавати зручні інструменти для дослідження великих обсягів даних, зокрема логів, подій, метрик і часових рядів. Kibana працює поверх Elasticsearch, використовуючи його індекси як джерело даних, тому вся аналітика, пошук і агрегації виконуються безпосередньо в кластері, а сама платформа відповідає за візуалізацію, навігацію та взаємодію з користувачем. (Рисунок 2.13).

Рисунок 2.13 – Платформа для роботи з даними «Kibana»

Однією з ключових можливостей Kibana є інструмент Discover, який дає змогу досліджувати дані у вигляді таблиць, виконувати пошукові запити, застосовувати фільтри та переглядати окремі документи. Це особливо корисно для аналізу логів, де важливо швидко знаходити потрібні події. Для побудови візуалізацій Kibana пропонує інструменти Lens і Visualize, які дозволяють створювати графіки, діаграми, heatmap, гістограми, геопросторові карти та інші типи панелей. Усі ці візуалізації можна об’єднувати в інтерактивні дашборди, які оновлюються в реальному часі та підтримують фільтри, drill-down і взаємодію між панелями. Завдяки цьому Kibana широко використовується для моніторингу інфраструктури, аналізу поведінки користувачів, безпекових подій і бізнес-метрик [10].

Kibana також включає спеціалізовані модулі для різних типів задач. Наприклад, Security і Observability надають готові інтерфейси для аналізу безпекових інцидентів, логів додатків, метрик системи та APM-трасувань. Модуль Machine Learning дозволяє виконувати автоматичне виявлення аномалій, прогнозування та класифікацію логів без необхідності створювати власні ML-моделі. Інструмент Canvas дає змогу створювати презентаційні дашборди з кастомним дизайном, а Maps – працювати з геоданими та будувати інтерактивні карти. Завдяки такій модульності Kibana може використовуватися як універсальна платформа для аналітики в різних доменах.

У сфері безпеки та адміністрування Kibana підтримує рольову модель доступу, інтеграцію з SSO, розмежування доступу до індексів і дашбордів, а

також можливість створювати окремі робочі простори (Spaces) для різних команд. Це робить платформу придатною для великих організацій, де важливо контролювати доступ до даних і розмежовувати середовища. Оскільки Kibana є частиною Elastic Stack, її можливості залежать від ліцензії: базові функції доступні у відкритій версії, тоді як розширені можливості безпеки, ML та інтеграції входять до комерційних підписок Elastic.

Загалом Kibana – це гнучка, масштабована та потужна платформа для візуалізації й аналізу даних, яка особливо ефективна в роботі з логами, подіями та часовими рядами. Її сила полягає в тісній інтеграції з Elasticsearch, широкому наборі інструментів для аналітики та можливості працювати з даними у реальному часі. Вона стала стандартом у сферах моніторингу, кібербезпеки, DevOps та операційної аналітики, забезпечуючи командам швидкий доступ до інформації та глибоке розуміння процесів у системах [11].

Приклад використання платформи на основі статистики з завантаження серверів (Рисунок 2.14), на основі статистики з продажів (Рисунок 2.15), статистика відвідування сайту (Рисунок 2.16).

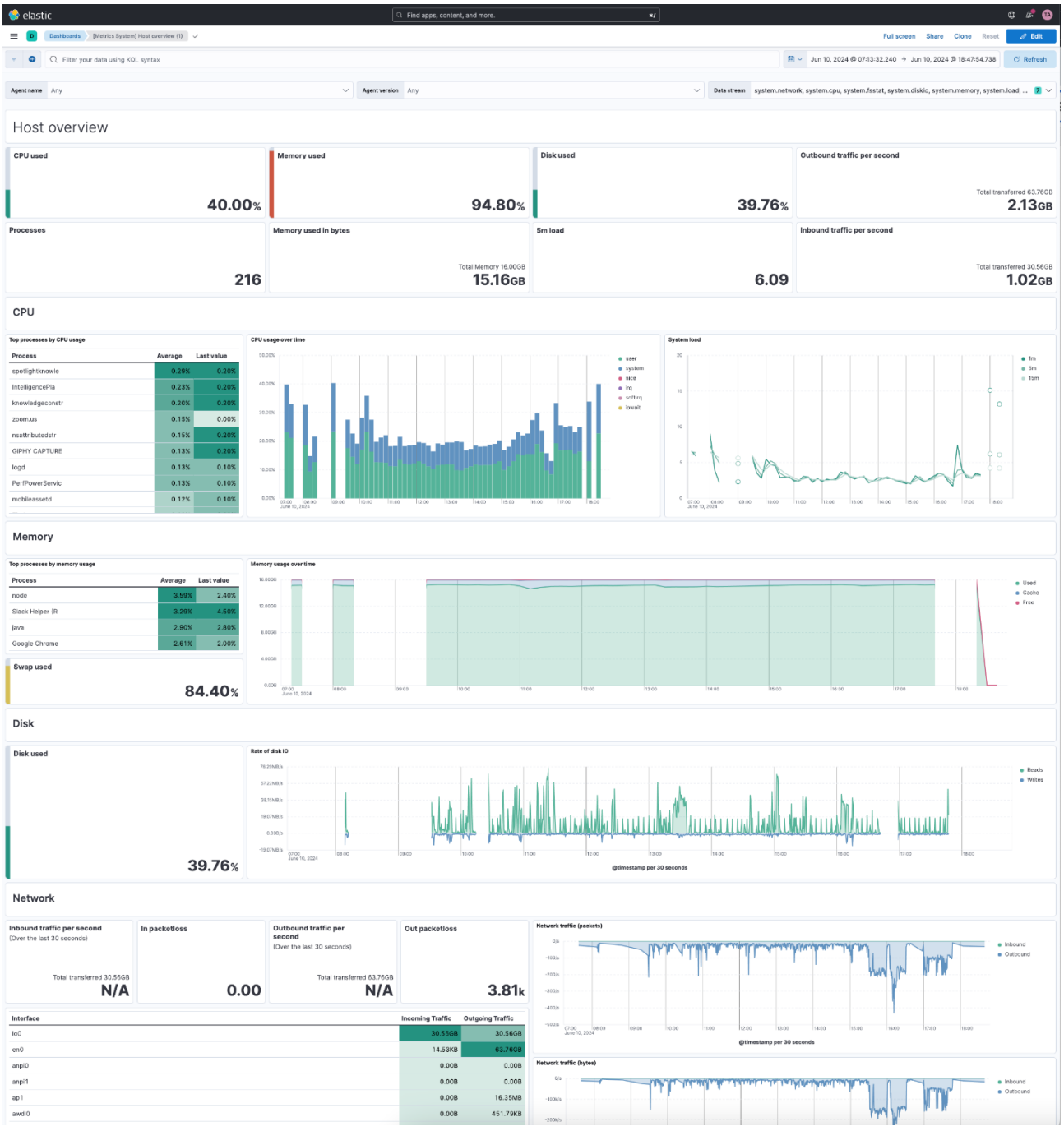


Рисунок 2.14 – Статистика завантаження серверів на платформі «Kibana»

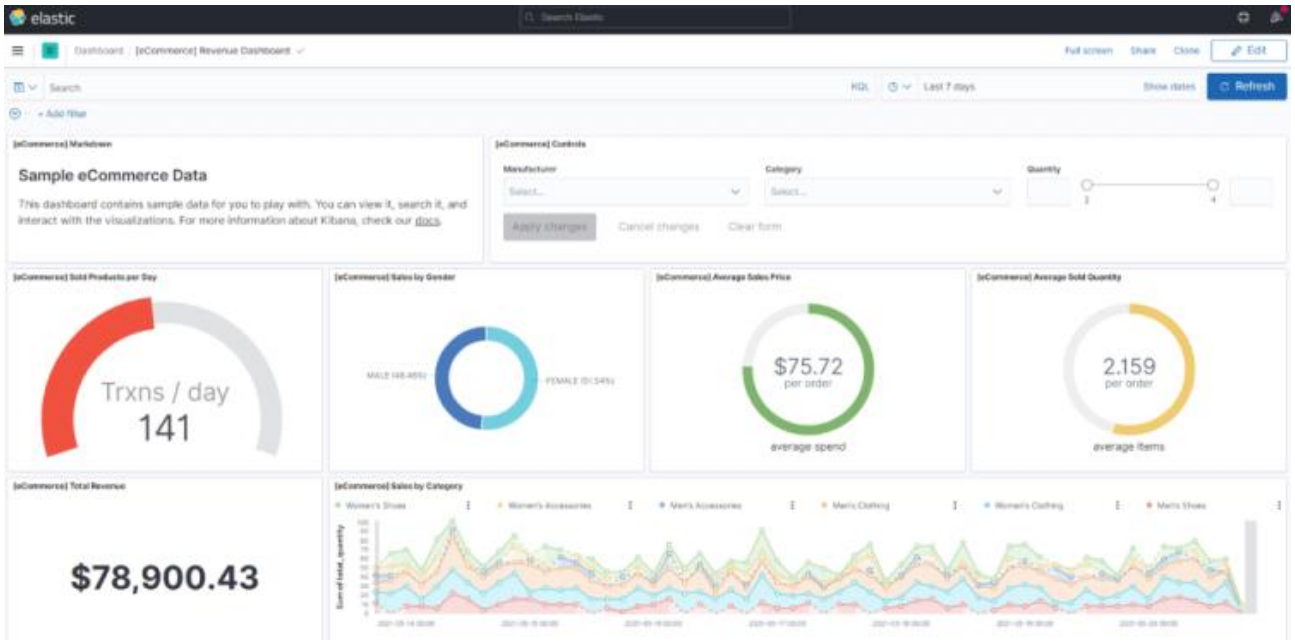


Рисунок 2.15 – Статистика з продажів на платформі «Kibana»

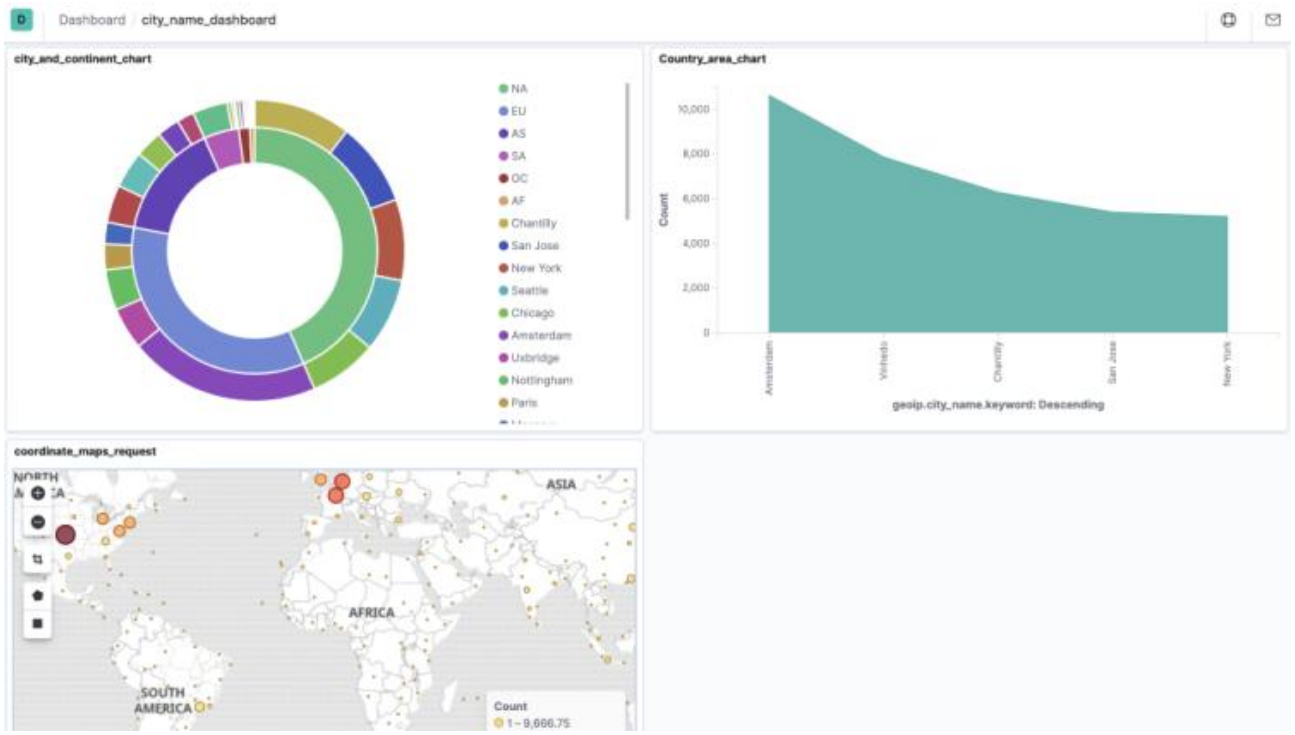


Рисунок 2.16 – Статистика з відвідування сайту на платформі «Kibana»

Переваги користування [12]:

− Глибока інтеграція з Elasticsearch. Kibana працює безпосередньо поверх Elasticsearch, використовуючи його індекси, агрегації та механізми пошуку. Це

забезпечує високу швидкість роботи з великими обсягами логів і подій у реальному часі.

− Потужні інструменти для аналізу логів і подій. Модуль Discover дозволяє швидко фільтрувати, шукати та переглядати окремі документи, що робить Kibana одним із найзручніших інструментів для log-аналізу.

− Розвинуті можливості візуалізації. Lens, Visualize та Canvas дають змогу створювати графіки, діаграми, heatmap, карти та презентаційні дашборди з високим рівнем кастомізації.

− Модульність і спеціалізовані рішення. Kibana включає готові інтерфейси для Observability (логи, метрики, APM), Security (SIEM), Machine Learning (аномалії, прогнозування), Maps (геодані).

− Робота в реальному часі. Дашборди оновлюються миттєво завдяки потоковому індексуванню Elasticsearch, що важливо для DevOps, SRE та SOC-команд.

− Гнучка система доступу та безпеки. Підтримка Spaces, SSO, рольової моделі та granular permissions дозволяє використовувати Kibana у великих організаціях.

− Масштабованість разом з Elastic Stack. Продуктивність Kibana зростає разом із масштабуванням Elasticsearch-кластера, що робить її придатною для petabyte-масштабів.

Недоліки користування:

− Залежність від Elasticsearch. Kibana не може працювати без Elasticsearch, тому її можливості повністю визначаються архітектурою та продуктивністю Elastic Stack.

− Високі вимоги до ресурсів при великих обсягах даних. При роботі з великими індексами Elasticsearch може споживати значні ресурси, що впливає на швидкість роботи Kibana.

− Обмежені можливості для класичної бізнес-аналітики. Kibana оптимізована для логів, подій і часових рядів, але менш підходить для фінансових звітів, складних BI-моделей або багатовимірної аналітики.

− Поріг входу для нових користувачів. Робота з KQL, Lucene Query Syntax, агрегаціями Elasticsearch і налаштуванням індексів вимагає технічних знань.

− Частина функцій доступна лише у платних ліцензіях Elastic. Machine Learning, розширені можливості безпеки, деякі інтеграції та аналітичні модулі доступні тільки у Gold/Platinum/Enterprise підписках.

− Складність адміністрування у великих середовищах. Потрібно налаштовувати індексні політики, ILM, безпеку, Spaces, ролі та оптимізацію кластерів, що вимагає досвідчених інженерів.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Опис проектних рішень та інструментів розробки

За результатами аналізу платформ аналогічного призначення, було визначено головні потреби, які не можуть дати інші платформи, а саме: доступність, простота інтерфейсу, прогнозування даних, open-source проєкт, залежність швидкодії від серверу де встановлена платформа.

Для розробки платформи було обрано інструмент проєктування інтерфейсів Figma, фронтенд-бібліотеку React, бібліотеку візуалізації D3.js, серверний фреймворк FastAPI, систему керування базами даних PostgreSQL, фреймворк для машинного навчання PyTorch, а також інструменти підтримки розробки – редактор коду Visual Studio Code та систему контролю версій Git.

Figma – це хмарний інструмент для проєктування інтерфейсів, прототипування та спільної роботи, розроблений компанією Figma, Inc. Заснування компанії відбулося у 2012 році, її співзасновниками є Dylan Field та Evan Wallace, а перша публічна версія продукту була представлена у 2016 році. На відміну від класичних десктопних редакторів, Figma з самого початку проєктувалася як веб-орієнтований сервіс із можливістю роботи у браузері, доповнений настільними застосунками (Рисунок 3.1).

Рисунок 3.1 – Інструмент для проєктування інтерфейсів «Figma»

Основна ціль Figma полягає в тому, щоб забезпечити єдине середовище для створення дизайну інтерфейсів та організації командної взаємодії у режимі реального часу. Інструмент покликаний усунути типові проблеми з обміном файлами, версіями та узгодженням макетів, замінюючи розрізнені локальні файли на централізовані хмарні документи, до яких одночасно можуть отримувати доступ дизайнери, розробники, менеджери та інші учасники процесу [13].

Функціональні можливості Figma охоплюють повноцінний векторний редактор із підтримкою фреймів, автолейаутів, стилів, сіток, масок та булевих операцій, що дозволяє створювати складні інтерфейсні макети й іконографіку. Важливим елементом є система компонентів і варіантів, яка забезпечує побудову масштабованих дизайн-систем: повторно використовувані елементи, стилі кольорів, типографіка, ефекти та токени дизайну можуть централізовано керуватися і застосовуватися в різних проєктах. Крім статичного дизайну, Figma підтримує інтерактивне прототипування: налаштування переходів між екранами, анімацій, інтерактивних компонентів і логіки поведінки прототипу, що дозволяє моделювати реальну взаємодію користувача з майбутнім застосунком. Хмарна природа інструменту забезпечує спільне редагування документів кількома користувачами одночасно, коментування безпосередньо на макеті, відстеження змін через історію версій та можливість надання доступу в режимах перегляду або редагування. Екосистема доповнюється плагінами та інтеграціями – від генерації контенту й роботи з іконками до зв’язку з Jira, Slack чи GitHub, а також окремим продуктом FigJam для фасилітації брейнштормінгів, діаграм і спільних сесій.

На практиці Figma використовується на всіх етапах розробки цифрових продуктів: від початкових ескізів і вайрфреймів до високодеталізованих макетів та інтерактивних прототипів, які застосовують для юзабіліті-тестування й демонстрації стейкхолдерам. Дизайнери створюють і підтримують дизайн-системи, розробники використовують специфікації, що автоматично генеруються Figma (розміри, відступи, кольори, стилі, CSS-фрагменти), для

верстки інтерфейсів, а продукт-менеджери та аналітики переглядають і коментують макети без потреби в окремому ПЗ. Завдяки роботі у браузері та хмарному зберіганню Figma зручна для розподілених команд і віддаленої співпраці [14].

Цільова аудиторія Figma включає UX/UI-дизайнерів, продукт-дизайнерів, фронтенд-розробників, продуктові команди, стартапи, а також освітні заклади, де інструмент використовується для навчання основам дизайну та проєктування інтерфейсів. Вона підходить як для невеликих команд, які цінують швидке розгортання та простоту, так і для великих організацій, що будують централізовані дизайн-системи і потребують керування доступами, масштабованих бібліотек компонентів та прозорої спільної роботи над інтерфейсами.

Переваги використання Figma [15]:

− Компонентна архітектура. Дозволяє інкапсулювати логіку й вигляд окремих частин інтерфейсу (графіки, таблиці, фільтри) та повторно використовувати їх у різних частинах системи.

− Хмарність і кросплатформеність. Робота напряму в браузері без складної інсталяції, однаково доступна на Windows, macOS, Linux і в браузері, що спрощує доступ до проєктів у розподілених командах.

− Спільна робота в реальному часі. Кілька користувачів можуть одночасно редагувати один документ, бачити курсори одне одного, залишати коментарі та обговорювати зміни без пересилання файлів і конфліктів версій.

− Підтримка дизайн-систем і повторного використання. Компоненти, варіанти, стилі й спільні бібліотеки дозволяють будувати централізовані дизайн-системи, зменшувати дублювання роботи та забезпечувати консистентність інтерфейсів у масштабі всього продукту чи організації.

− Інтерактивне прототипування в одному середовищі. Створення прототипів без експорту в інші інструменти – переходи, анімації, інтерактивні елементи реалізуються прямо в макеті, що прискорює тестування сценаріїв і узгодження рішень.

− Зручна передача дизайну розробникам. Автоматичне надання специфікацій (розміри, відступи, кольори, стилі, CSS-фрагменти), експорт ресурсів і режим перегляду спрощують імплементацію інтерфейсів і зменшують кількість ручних помилок.

− Багата екосистема плагінів та інтеграцій. Плагіни для генерації контенту, роботи з іконками, локалізації, перевірки доступності, інтеграцій з Jira, Slack, GitHub та іншими сервісами дозволяють адаптувати Figma до конкретних процесів команди.

− Підтримка колективних сесій і фасилітації (FigJam). Окремий інструмент для брейнштормінгів, діаграм, воркшопів і карт користувацьких сценаріїв допомагає об’єднати етапи дослідження, проєктування й обговорення в єдиному середовищі.

− Низький поріг входу та зручність навчання. Інтуїтивний інтерфейс, велика кількість шаблонів, навчальних матеріалів і спільнота роблять Figma доступною як для початківців, так і для досвідчених дизайнерів.

React – це відкрита бібліотека JavaScript для побудови інтерфейсів користувача, яка була створена інженером Джорданом Волком у межах Facebook і вперше опублікована як проєкт з відкритим кодом у 2013 році. Надалі розвитком React опікується Meta (раніше Facebook) разом із широкою спільнотою розробників; бібліотека поширюється під ліцензією MIT і стала однією з найпопулярніших технологій для створення сучасних веб-застосунків (Рисунок 3.2).

Рисунок 3.2 – Відкрита бібліотека JavaScript «React»

Ідея React виникла в інженерній команді Facebook як відповідь на складнощі з підтримкою динамічних інтерфейсів (зокрема стрічки новин), і перша публічна версія з'явилася у 2013 році; автором початкової реалізації вважають Jordan Walke, а подальший розвиток веде Meta разом зі спільнотою розробників [16].

Головна мета React – спростити створення складних, динамічних інтерфейсів шляхом компонентного підходу та декларативного опису UI. React прагне відокремити логіку від представлення, зробити оновлення інтерфейсу передбачуваними та ефективними (через концепцію віртуального DOM), що дозволяє розробникам зосередитися на моделюванні стану застосунку, а не на ручних маніпуляціях DOM.

React надає набір базових механізмів і патернів для побудови інтерфейсів: компонентну модель (функціональні та класові компоненти), JSX-синтаксис для декларативного опису розмітки, систему пропсів для передачі даних у компоненти та локальний стан для управління внутрішнім станом компонентів. Ключовим елементом є віртуальний DOM – проміжне представлення реального DOM, яке дозволяє ефективно обчислювати мінімальні зміни й застосовувати їх пакетно, зменшуючи кількість дорогих операцій рендерингу. З появою хуків (Hooks) React отримав механізм для організації логіки стану та побічних ефектів у функціональних компонентах (useState, useEffect, useMemo, useCallback, useContext тощо), що спростило повторне використання логіки та тестування. Додаткові можливості включають контекст (Context API) для передачі даних через дерево компонентів без проп-пробросу, портали для рендерингу елементів поза основним DOM-деревом, фрагменти для групування елементів без зайвих вузлів у DOM, а також механізми для серверного рендерингу (SSR) і гідратації, що важливо для SEO та швидкого першого відображення сторінки. React також підтримує роботу з рефами (refs) для прямого доступу до DOM-елементів або екземплярів компонентів, а екосистема

пропонує інструменти для маршрутизації (React Router), управління глобальним станом (Redux, Zustand, Recoil), форм (Formik, React Hook Form) та тестування (Jest, React Testing Library) [17].

У проєктах React зазвичай організовують UI як набір дрібних, ізольованих компонентів: від простих візуальних елементів (кнопка, інпут) до складних контейнерів (дашборд, таблиця з фільтрами). Компоненти комбінуються у композиції, де дані передаються зверху вниз через пропси, а локальний або глобальний стан керує поведінкою. Для інтеграції з сервером застосовують HTTP-клієнти (fetch, Axios) або GraphQL-клієнти (Apollo), а для оптимізації рендерингу використовують мемоізацію компонентів і селекторів стану. У поєднанні з D3.js React часто відповідає за життєвий цикл та структуру UI, тоді як D3 виконує низькорівневий рендеринг складних візуалізацій; можливі підходи – керувати DOM-вставками D3 всередині життєвого циклу React-компонента або використовувати D3 лише для обчислення геометрії й масштабів, а рендерити через React. Для розгортання застосунків застосовують інструменти збірки (Vite, Webpack, Create React App), CI/CD-конвеєри та контейнеризацію.

React орієнтований на веб-розробників і команди, які створюють інтерфейси з високою динамікою даних, складною логікою відображення або потребою в масштабованій архітектурі компонентів – це корпоративні SPA, аналітичні дашборди, інструменти для редагування контенту, інтерфейси для мобільних застосунків (через React Native) та інші інтерактивні продукти. React підходить як для невеликих команд, що прагнуть швидко прототипувати, так і для великих організацій, які потребують зрілої екосистеми, стандартів розробки та інструментів для підтримки життєвого циклу проєкту.

Переваги використання React [18]:

− Компонентна архітектура. Дозволяє інкапсулювати логіку й вигляд окремих частин інтерфейсу (графіки, таблиці, фільтри) та повторно використовувати їх у різних частинах системи.

− Висока продуктивність. Завдяки віртуальному DOM зменшується кількість операцій з реальним DOM, що важливо для динамічних дашбордів і частих оновлень даних.

− Розвинена екосистема. Велика кількість бібліотек (React Router, state-менеджери, форми тощо) спрощує реалізацію складних інтерфейсних сценаріїв.

− Сильна спільнота та документація. Полегшує навчання, пошук рішень і забезпечує довгострокову підтримку технології.

− Добра інтеграція з D3.js. Дає змогу поєднати декларативний UI з низькорівневими візуалізаціями.

D3.js (Data-Driven Documents) – це відкрита JavaScript-бібліотека для створення інтерактивних візуалізацій даних у веб-браузері. Вона надає набір низькорівневих інструментів для зв’язування даних із елементами документа (DOM) та для програмного керування їхнім графічним представленням через SVG, Canvas або HTML. D3 не нав’язує готові шаблони візуалізацій, натомість дає розробникові повний контроль над перетворенням даних у візуальні форми (Рисунок 3.3).

Рисунок 3.3 – Відкрита бібліотека JavaScript «D3.js»

Ідея та перша реалізація D3 належать Майку Бостоку (Mike Bostock) і його колегам; ключова публікація «Data-Driven Documents» з'явилася у 2011 році. З того часу D3 розвивається як проєкт з відкритим кодом і широко використовується в академічних, журналістських та інженерних проєктах візуалізації [19].

Головна мета D3 – забезпечити «зв'язування даних з документом»: дозволити розробнику безпосередньо прив'язувати набори даних до DOM-елементів і визначати, як ці елементи мають створюватися, оновлюватися та видалятися у відповідь на зміни даних. Такий підхід робить візуалізації максимально гнучкими, дозволяє реалізувати нетипові графіки та інтерактивні інтерфейси, а також інтегрувати візуалізації з інструментами розробки браузера.

D3 застосовують у двох основних ролях: як повноцінний рендер-движок для кастомних візуалізацій і як набір утиліт для обчислення геометрії та логіки візуалізації, залишаючи рендеринг за іншими бібліотеками. У проєктах на React або інших UI-фреймворках поширені підходи, коли D3 відповідає за обчислення шкал, осей і координат, а React рендерить елементи, або навпаки – D3 безпосередньо маніпулює SVG у межах життєвого циклу компонентів. D3 широко використовується для створення інтерактивних дашбордів, наукових візуалізацій, інфографіки в журналістиці, інструментів для дослідження даних (exploratory data analysis) та кастомних UI-компонентів у бізнес-застосунках.

D3 орієнтований на розробників візуалізацій, фронтенд-інженерів, дослідників даних, візуалізаторів у журналістиці та науковців, які потребують високого ступеня контролю над відображенням даних. Це інструмент для тих, кому потрібна гнучкість і точність у візуалізації, а не готові «чарт-бібліотеки» з обмеженим набором графіків. D3 підходить як для індивідуальних прототипів і дослідницьких робіт, так і для промислових продуктів, де потрібні нестандартні або високопродуктивні візуалізації.

D3.js надає широкий набір функціональних можливостей для перетворення даних у візуальні представлення: бібліотека дозволяє зв'язувати

масиви даних із DOM-елементами за патерном enter-update-exit, обчислювати та застосовувати шкали (лінійні, логарифмічні, часові, категоріальні) і генерувати осі з форматуванням міток, створювати геометричні примітиви та складні форми (лінії, області, арки, стеки, кругові діаграми) за допомогою генераторів, виконувати макетування складних структур (force-directed графи, дерева, упаковки, контури тощо), керувати переходами й анімаціями для плавного відображення змін даних, обробляти події користувача та реалізовувати інтерактивні поведінки (масштабування, панорамування, виділення, перетягування, brush), працювати з геоданими і проєкціями для побудови карт, завантажувати й парсити різні формати даних (CSV, JSON, GeoJSON) та застосовувати обчислювальні утиліти (інтерполяції, форматування, агрегації); при цьому обчислювальні можливості D3 можна використовувати разом із різними механізмами рендерингу (SVG, Canvas, WebGL), що дає змогу поєднувати точні візуалізації з високою продуктивністю при великій кількості елементів [20].

D3.js – це потужний, низькорівневий набір інструментів для перетворення даних у візуальні представлення з високим ступенем контролю над рендерингом і взаємодією. Його сила – у гнучкості та можливості реалізувати практично будь-яку візуалізацію, але це також означає вищий поріг входу порівняно з готовими бібліотеками. D3 найкраще підходить для проєктів, де потрібні кастомні, інтерактивні та точні візуалізації даних.

Переваги використання D3.js [21]:

− Низькорівневий контроль візуалізації. Дає змогу створювати кастомні графіки будь-якої складності, а не обмежуватися фіксованим набором типових діаграм.

− Гнучкість для аналітики. Підходить для візуалізації часових рядів, багатовимірних даних, результатів ML-моделей, кореляцій та нестандартних метрик.

− Підтримка анімацій та інтерактивності. Забезпечує плавні переходи, наведення, масштабування, панорамування та інші інтерактивні сценарії.

− Робота зі шкалами та осями. Спрощує побудову складних координатних систем, нормалізацію та форматування даних.
− Інтегрованість з React: може використовуватися всередині React-компонентів як рендер-движок для графіків.

FastAPI – це сучасний високопродуктивний веб-фреймворк для побудови HTTP-API на мові Python, створений Себастьяном Раміресом і вперше опублікований у 2018 році; проєкт швидко здобув популярність завдяки поєднанню простоти розробки та високої продуктивності (Рисунок 3.4).

Рисунок 3.4 – Фреймворк «FastAPI»

FastAPI розроблено як легковагове, типобезпечне рішення, яке використовує стандартні підказки типів Python (type hints) і бібліотеку Pydantic для валідації та серіалізації даних, а також базується на асинхронному стандарті ASGI, що дозволяє досягати продуктивності, порівнянної з Node.js і Go у типових API-навантаженнях [22].

Головна ціль FastAPI – спростити і прискорити розробку надійних, документованих і масштабованих API шляхом автоматичної генерації OpenAPI-специфікацій і інтерактивної документації (Swagger UI, ReDoc), забезпечення суворої валідації вхідних і вихідних даних та мінімізації шаблонного коду. Це дозволяє розробникам фокусуватися на бізнес-логіці, а не на рутинних аспектах маршрутизації, валідації та документування інтерфейсів.

Функціональні можливості FastAPI охоплюють: декларативну маршрутизацію HTTP-ендпоїнтів з підтримкою всіх стандартних методів (GET,

POST, PUT, DELETE тощо); автоматичну валідацію та серіалізацію через Pydantic-моделі; обробку параметрів шляху, запиту, заголовків і тіла запиту; асинхронні обробники для неблокуючих викликів до баз даних або зовнішніх сервісів; механізми dependency injection для організації залежностей і повторного використання логіки; підтримку middleware, background-tasks і WebSocket; а також вбудовану генерацію OpenAPI/JSON Schema і інтерактивних UI для тестування API, що значно полегшує інтеграцію та тестування сервісів.

На практиці FastAPI використовується як тонкий, продуктивний шар між клієнтськими застосунками та бізнес-логікою або моделями машинного навчання: його застосовують для побудови REST/HTTP-сервісів, мікросервісів, API-шлюзів, бекендів для SPA та мобільних додатків, а також для розгортання ML-інференсу (через інтеграцію з бібліотеками на кшталт PyTorch або TensorFlow). Асинхронна природа та мінімалістичний синтаксис роблять FastAPI зручним для швидкого прототипування й одночасно придатним для продакшн-середовищ з високими навантаженнями [23].

FastAPI орієнтований на Python-розробників, інженерів даних і команд, які потребують швидкої розробки типобезпечних API з автоматичною документацією та високою пропускною здатністю. Він підходить як для невеликих стартап-проєктів, де важлива швидкість розробки, так і для великих команд, що будують мікросервісну архітектуру або інтегрують моделі машинного навчання у веб-сервіси; завдяки сумісності з існуючими інструментами Python-екосистеми FastAPI легко інтегрується у вже наявні пайплайни розробки та деплою.

Переваги використання FastAPI [24]:

− Висока продуктивність. Асинхронна модель виконання дозволяє обробляти велику кількість паралельних запитів, що критично для аналітичних систем.

− Сильна типізація та валідація. Pydantic-моделі гарантують коректність вхідних і вихідних даних, спрощуючи підтримку API.

− Автоматична документація. Генерує OpenAPI-специфікації та інтерактивні інтерфейси (Swagger UI, ReDoc) без додаткових зусиль.
− Природна інтеграція з Python-ML. Зручно викликати PyTorch-моделі без «містків» на інші мови чи середовища.
− Зручність розробки. Мінімальний шаблонний код, зрозумілий синтаксис і швидкий старт проєкту.

PostgreSQL – зріла, багатофункціональна та розширювана реляційна СУБД з акцентом на коректність, стандарти SQL і можливість адаптації під різні доменні задачі; її архітектура та екосистема роблять її надійним вибором як для транзакційних систем, так і для аналітичних або геопросторових застосунків (Рисунок 3.5).

Рисунок 3.5 – Реляційна СУБД «PostgreSQL»

PostgreSQL походить від академічного проєкту POSTGRES, що розвивався в Університеті Каліфорнії в Берклі під керівництвом професора Майкла Стоунбрейкера; сучасна гілка PostgreSQL як відкритого проєкту отримала першу стабільну публічну версію в середині 1990-х років і офіційно вважається випущеною у 1996 році. Розвитком і підтримкою системи нині опікується PostgreSQL Global Development Group, а офіційна документація систематично фіксує еволюцію функціоналу та архітектурні рішення проєкту [25].

Головна мета PostgreSQL – надати потужну, надійну та розширювану реляційну систему керування базами даних, яка відповідає стандартам SQL,

забезпечує транзакційну коректність (ACID) і водночас дозволяє розширювати функціонал через модулі та розширення. Проєкт поєднує класичну реляційну модель з можливістю роботи з напівструктурованими даними та спеціалізованими типами (наприклад, геопросторовими), що робить його універсальним інструментом для широкого спектра застосувань.

PostgreSQL надає повний набір можливостей реляційної СУБД: створення та керування схемами даних (таблиці, індекси, обмеження цілісності), виконання складних SQL-запитів із JOIN, підзапитами, віконними функціями та CTE; підтримку транзакцій із різними рівнями ізоляції та гарантіями ACID; механізми індексації (B-tree, GiST, GIN, BRIN тощо) для оптимізації запитів; тригери, збережені процедури та користувацькі функції для реалізації бізнес-логіки на рівні БД. Крім того, PostgreSQL має розширену підтримку типів даних: JSON/JSONB для ефективної роботи з напівструктурованими документами, масиви, користувацькі типи, а також розширення для геоданих (PostGIS) і повнотекстового пошуку; система також включає засоби для планування та оптимізації запитів, матеріалізовані подання, механізми бекапу/відновлення і моніторингу продуктивності.

У практичних проєктах PostgreSQL застосовують як основне сховище для транзакційних і аналітичних навантажень: зберігання бізнес-даних, журналів подій, конфігурацій, результатів обчислень і метаданих. Для продакшн-розгортань використовують реплікацію (streaming replication), логічну реплікацію, шарування на рівні застосунку, connection pooling (PgBouncer, Pgpool), налаштування індексів і матеріалізованих подань для пришвидшення аналітичних запитів, а також інтеграцію з інструментами моніторингу й автоматизації розгортання; у геопросторових задачах PostgreSQL із PostGIS часто виступає як основа ГІС-рішень [26].

PostgreSQL орієнтований на широкий спектр користувачів: від розробників і стартапів, яким потрібна надійна безкоштовна СУБД, до великих підприємств і державних установ, що потребують масштабованих, транзакційно-коректних рішень з можливістю розширення функціоналу. Він

підходить для бекенд-інженерів, інженерів даних, аналітиків, розробників ГІС-систем, фінансових застосунків та будь-яких проєктів, де важлива цілісність даних, гнучкість типів даних і можливість тонкого налаштування продуктивності.

Переваги використання PostgreSQL [27]:

− Надійність і транзакційність. Повна підтримка ACID гарантує цілісність даних при одночасних операціях.

− Потужний SQL і розширення. Підтримка складних запитів, індексів, тригерів, збережених процедур та розширень (зокрема PostGIS).

− Підтримка JSON/JSONB. Дозволяє комбінувати реляційні та напівструктуровані дані в одній системі.

− Масштабованість і продуктивність. Можливість реплікації, оптимізації запитів і роботи з великими обсягами даних.

− Відсутність залежності від вендора. Відкритий код і активна спільнота знижують ризики прив’язки до конкретного постачальника.

PyTorch – це відкритий фреймворк для машинного навчання та глибинного навчання, розроблений командою Facebook AI Research (FAIR) і вперше опублікований у 2016 році. Проєкт швидко здобув популярність у науковій спільноті та індустрії завдяки поєднанню інтуїтивного Python-інтерфейсу, гнучкості для досліджень і можливостей для промислового застосування. Надалі PyTorch розвивається спільнотою розробників і підтримується організаціями, що використовують його у дослідженнях і продуктах (Рисунок 3.6).

Рисунок 3.6 – Фреймворк «PyTorch»

Головна мета PyTorch – забезпечити зручний, гнучкий і продуктивний інструментарій для побудови, навчання та розгортання нейронних мереж. PyTorch орієнтований на швидке прототипування моделей у дослідницьких задачах, одночасно надаючи механізми для оптимізації й перенесення моделей у продакшн-середовище. Фреймворк прагне знизити «тертя» між експериментом і виробництвом, дозволяючи розробникам використовувати один і той самий код для досліджень і сервісів [28].

PyTorch надає повний набір функціональних можливостей для сучасного ML-розробника. Тензори і обчислення – базовий тип даних Tensor підтримує операції на CPU і GPU, ефективні алгебраїчні операції та широкі можливості для лінійної алгебри. Автоматичне диференціювання (autograd) автоматично обчислює градієнти для складних обчислювальних графів, що спрощує реалізацію алгоритмів оптимізації. Модульна нейронна абстракція (torch.nn) дозволяє будувати шари, модулі та складні архітектури з повторним використанням компонентів. Оптимізатори і функції втрат (torch.optim) забезпечують стандартні та просунуті алгоритми оптимізації, регуляризації та scheduler-и для навчання. Завантаження даних (torch.utils.data) включає Dataset і DataLoader з підтримкою батчування, паралельного препроцесингу та кастомних трансформацій. Підтримка GPU і CUDA дає змогу прискорювати навчання й інференс на апаратному прискоренні. Розподілене навчання і torch.distributed дозволяють масштабувати тренування на кілька GPU і вузлів. Інструменти для продакшну включають TorchScript і JIT-компіляцію для трансляції моделей у статичні графи, TorchServe для розгортання сервісів інференсу, а також інтеграції з контейнерами й хмарними платформами. Екосистема доповнюється бібліотеками для комп’ютерного зору, NLP і аудіо (TorchVision, TorchText, Torchaudio), а також інструментами для тренування, логування та оптимізації (Lightning, Optuna, ONNX-експорт).

На практиці PyTorch застосовують у двох основних сценаріях: дослідження і прототипування та промислове розгортання. У дослідницькому середовищі PyTorch цінується за динамічні обчислювальні графи, які

дозволяють використовувати звичайні Python-конструкції (цикли, умовні оператори) під час побудови моделі, що спрощує налагодження і експерименти. Для продакшну розробники використовують TorchScript або експортують моделі в ONNX для оптимізованого виконання; також поширене розгортання через TorchServe або контейнеризацію з Kubernetes. PyTorch інтегрують у пайплайни підготовки даних (Pandas, NumPy), обробки великих датасетів (Dataloader, Datasets), а також у системи CI/CD для автоматичного тренування, тестування і деплою моделей.

PyTorch призначений для широкого кола користувачів: дослідників і науковців, які експериментують з архітектурами нейронних мереж; інженерів машинного навчання і ML-інженерів, що будують і оптимізують моделі для продуктів; інженерів даних і розробників, які інтегрують моделі в бекенд-сервіси; а також освітян і студентів, що вивчають глибинне навчання. PyTorch підходить як для швидкого прототипування в лабораторіях, так і для масштабованих виробничих систем у компаніях різного розміру [29].

PyTorch поєднує інтуїтивний Python-інтерфейс і потужні механізми для обчислень на GPU, автоматичного диференціювання, модульного побудування мереж і розгортання в продакшн. Його гнучкість робить його фаворитом у дослідницькій спільноті, а розвинена екосистема і інструменти для продакшну забезпечують шлях від експерименту до сервісу без значних переписувань коду.

Переваги використання PyTorch [30]:

− Динамічні обчислювальні графи. Спрощують експерименти з архітектурами моделей, налагодження та дослідницьку роботу.

− Підтримка GPU: пришвидшує навчання й інференс моделей, що важливо для реальних даних і складних архітектур.

− Зручний Python-інтерфейс. Природно інтегрується з рештою Python-екосистеми (NumPy, Pandas тощо).

− Багата екосистема. Додаткові бібліотеки (TorchVision, Lightning та ін.) прискорюють розробку та експерименти.

− Добра інтеграція з FastAPI. Дозволяє будувати ML-сервіси для інференсу в реальному часі.

Visual Studio Code – кросплатформений редактор коду, розроблений компанією Microsoft і вперше випущений у квітні 2015 року. Проєкт швидко перейшов у модель відкритого розвитку: ядро редактора розміщено на GitHub, а дистрибутив поширюється як безкоштовний пропрієтарний продукт із великою кількістю розширень. Архітектура VS Code побудована на Electron і використовує веб-технології (TypeScript, JavaScript, HTML, CSS), що забезпечує однаковий інтерфейс і поведінку на Windows, macOS і Linux (Рисунок 3.7).

Рисунок 3.7 – Редактор коду «Visual Studio Code»

Головна мета Visual Studio Code – надати легковаговий, швидкий і розширюваний інструмент для розробки програмного забезпечення, який поєднує зручність текстового редактора з можливостями IDE. VS Code прагне забезпечити баланс між продуктивністю та функціональністю: бути достатньо простим для швидкого редагування файлів і одночасно достатньо потужним для налагодження, інтеграції з системами контролю версій і роботою з контейнерами та віддаленими середовищами [31].

Visual Studio Code надає повний набір можливостей для сучасної розробки. Редактор підтримує підсвічування синтаксису та розумне автодоповнення (IntelliSense) на основі аналізу коду й типів, що підвищує

швидкість написання та зменшує кількість помилок. Вбудований механізм налагодження дозволяє запускати програми, ставити точки зупину, переглядати стек викликів і змінні без переходу в інші інструменти. Інтегрований термінал дає змогу виконувати командні рядки, запускати скрипти та керувати середовищем розробки без виходу з редактора.

Система розширень є ключовою функцією: через Marketplace можна додавати підтримку мов, лінтерів, форматерів, інструментів для роботи з Docker, Kubernetes, базами даних, хмарними сервісами та інструментами CI/CD. Розширення також забезпечують інтеграцію з системами контролю версій, інструментами для тестування та профілювання. VS Code підтримує роботу з проектами на різних мовах завдяки Language Server Protocol (LSP), що дозволяє однаково підключати інтелектуальні сервіси для автодоповнення, переходу до визначення, рефакторингу та діагностики.

Інструменти для роботи з кодом включають рефакторинг, швидкий перехід до символів і файлів, багатофайловий пошук і заміну, фрагменти коду (snippets) та можливість налаштування робочих просторів. Для командної роботи VS Code має вбудовану підтримку Git: перегляд змін, створення комітів, робота з гілками та вирішення конфліктів прямо в інтерфейсі. Додаткові можливості включають віддалене редагування через SSH/WSL/Remote Containers, інтеграцію з контейнерами Docker, підтримку Dev Containers для відтворюваних середовищ розробки, а також інструменти для роботи з Jupyter Notebook і інтерактивною розробкою даних.

У практиці VS Code використовується як основне робоче середовище для розробки фронтенду (React, Angular, Vue), бекенду (Node.js, Python, Go, .NET), для створення скриптів, автоматизації та роботи з інфраструктурою. Розробники налаштовують робочі простори з набором розширень і конфігурацій (форматери, лінтери, налаштування запуску), використовують інтегрований термінал для запуску локальних серверів і тестів, а також підключаються до віддалених середовищ через Remote-функціональність. У командних проєктах VS Code часто поєднують із системами CI/CD,

контейнеризацією та інструментами моніторингу, що дозволяє створити відтворюване середовище розробки та спростити онбординг нових учасників [32].

Visual Studio Code орієнтований на широкий спектр користувачів: індивідуальних розробників, невеликі команди продукту, інженерів DevOps, інженерів даних і студентів. Він підходить як для швидкого прототипування та редагування файлів, так і для побудови складних проєктів завдяки розширюваності та інтеграції з інструментами розробки. VS Code корисний для тих, хто цінує легковаговість редактора, але потребує можливостей IDE без важкої конфігурації та з можливістю гнучкого налаштування під конкретні робочі процеси.

Visual Studio Code поєднує простоту текстового редактора з потужністю IDE через багату екосистему розширень, інтелектуальні можливості редагування, налагодження та інтеграцію з інструментами розробки. Це універсальний інструмент, який підходить для більшості сучасних робочих навантажень у розробці програмного забезпечення і значно підвищує продуктивність як індивідуальних розробників, так і команд.

Переваги використання Visual Studio Code [33]:

− Інтелектуальне доповнення коду (IntelliSense). Автодоповнення, підказки по сигнатурам функцій і швидкий доступ до визначень, що пришвидшує написання коду і зменшує кількість помилок.

− Швидка навігація по проєкту. Перехід до визначень, пошук символів і швидке відкриття файлів роблять робочий процес більш плавним.

− Велика екосистема розширень. Тисячі плагінів для мов, лінтерів, форматерів, Docker, Kubernetes та інших інструментів дозволяють підлаштувати редактор під будь-які задачі.

− Вбудоване налагодження. Підтримка breakpoint, watch, call stack і інтеграція з віддаленим дебагом дозволяє налагоджувати без переходу в інші інструменти.

− Інтеграція з Git і контролем версій. Зручні панелі для commit, branch, diff і merge прямо в редакторі.

– Кросплатформеність. Однаковий інтерфейс і конфігурації на Windows, macOS і Linux, що спрощує роботу в команді з різними ОС.
– Легкість і продуктивність. Швидкий запуск і низьке споживання ресурсів у порівнянні з важкими IDE. Можливість масштабувати функціональність через розширення.
– Висока налаштовуваність. Теми, шорткати, налаштування робочого простору і конфігурації під конкретні проєкти.
– Активна спільнота і документація. Регулярні оновлення, офіційні гіди та велика кількість прикладів і туторіалів.

Git – це розподілена система контролю версій (DVCS), створена Лінусом Торвальдсом у 2005 році як відповідь на потребу в надійному, швидкому та розподіленому інструменті для керування змінами в коді ядра Linux; подальній підтримці та розвитку проєкту сприяє широка спільнота розробників і офіційна документація проєкту (Рисунок 3.8).

Рисунок 3.8 – Розподілена система контролю версій «Git»

Головна мета Git – забезпечити ефективне відстеження змін у файлах проєкту, підтримку паралельної розробки через гілки, швидкі локальні операції та надійну синхронізацію з віддаленими репозиторіями. Git проєктувався так, щоб кожен клон репозиторію містив повну історію проєкту, що підвищує стійкість до збоїв мережі та серверів і дає змогу виконувати більшість операцій локально без доступу до центрального сервера [34].

Git надає повний набір механізмів для версіонування та управління кодом: створення та фіксація змін (commits) зі знімками стану проєкту; робота з гілками (branching) і злиттями (merging) для організації паралельної розробки; операції синхронізації з віддаленими репозиторіями (clone, fetch, pull, push); створення тегів (tags) для маркування релізів; інструменти для інтерактивного редагування історії (rebase, cherry-pick, revert) та тимчасового збереження незавершених змін (stash). Додаткові можливості включають механізми діагностики та відновлення (reflog, bisect), хуки для автоматизації робочих процесів на клієнті або сервері, підтримку підмодулів (submodule) для включення зовнішніх репозиторіїв, робочі дерева (worktree) для одночасної роботи з кількома гілками, а також розширення для роботи з великими файлами (Git LFS). Архітектурно Git зберігає дані як об’єкти (blobs, trees, commits, tags) і використовує ефективні механізми дельта-схем та індексації, що забезпечує високу швидкість операцій навіть для великих кодових баз.

У практиці Git застосовується як основний інструмент для командної розробки: розробники створюють гілки для фіч, багфіксів або експериментів, виконують локальні коміти та регулярно синхронізують роботу з віддаленим репозиторієм через push/pull; процеси код-рев’ю організовуються через pull-/merge-request-флоу на платформах типу GitHub або GitLab, а CI/CD-системи тригеряться на події в репозиторії для автоматичного тестування, білду та деплою. У продакшн-середовищах застосовують практики connection-pooling для репозиторіїв, політики гілкування (Git Flow, GitHub Flow, trunk-based development), автоматизацію через хуки та інтеграцію з інструментами для управління релізами й відстеженням задач. Git також використовується поза сферою розробки ПЗ – для версіонування конфігурацій, документів, інфраструктурного коду та даних, де важлива історія змін і можливість відкотитися до попередніх станів [35].

Git призначений для програмістів, команд розробки, DevOps-інженерів, інженерів даних і будь-яких фахівців, які потребують надійного контролю версій і координації спільної роботи над артефактами. Він підходить як для

індивідуальних проєктів і стартапів, так і для великих організацій з розподіленими командами та складними процесами реліз-менеджменту.

Переваги використання Git [36]:

− Повна історія змін. Дозволяє відстежувати еволюцію коду, повертатися до будь-якої версії, аналізувати зміни.

− Зручна робота з гілками. Підтримка branching/merging спрощує паралельну розробку, експерименти та код-рев'ю.

− Підтримка віддалених репозиторіїв. GitHub, GitLab, Bitbucket забезпечують командну роботу, CI/CD та управління задачами.

− Інтеграція з IDE. У Visual Studio Code Git є частиною повсякденного робочого процесу.

− Надійність і швидкодія. Локальні операції працюють швидко й не залежать від мережі.

3.2. Опис обраного підходу до створення програмного забезпечення

В результаті розгляду та аналізу вище перелічених засобів та інструментів розробки, було вирішено обрати гнучку модель (Agile model), як підхід до розробки платформи для візуалізації масивів даних з використанням сучасних методів машинного навчання.

Agile – це підхід до розробки програмного забезпечення, що ставить у центрі швидку доставку цінності користувачу, адаптивність до змін і тісну співпрацю між зацікавленими сторонами. Agile базується на декларативних принципах і цінностях, які підкреслюють ітеративну роботу, короткі цикли поставки, постійний зворотний зв'язок і прагнення до безперервного покращення процесів і продукту.

Agile-модель поділяє завдання на часові рамки, щоб забезпечити конкретну функціональність для релізу. Кожна збірка є інкрементною з точки зору функціональності, а фінальна збірка містить всі атрибути. Поділ всього проекту на невеликі частини допомагає мінімізувати проектний ризик і загальний час реалізації проекту [37] (Рисунок 3.9).

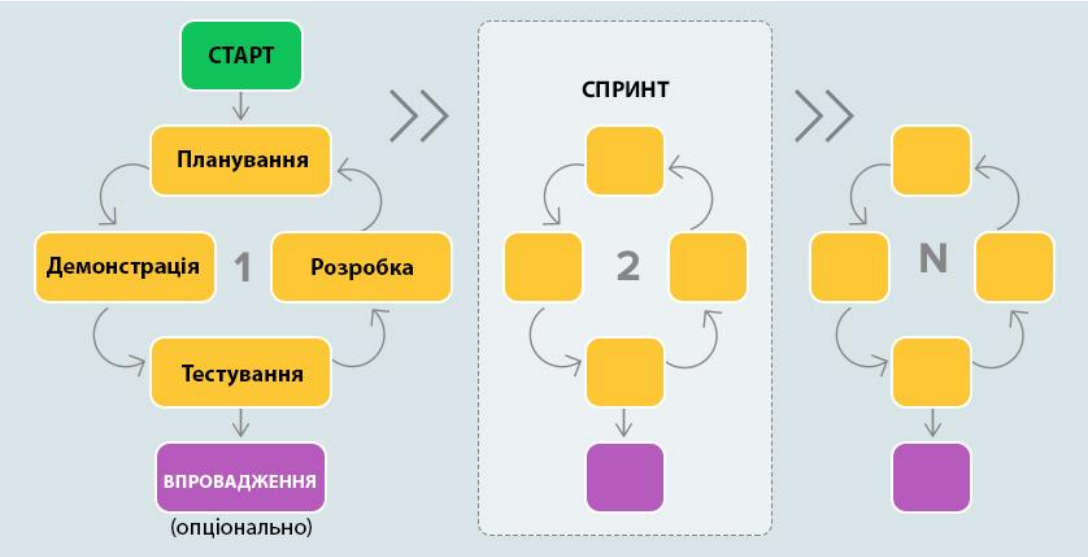


Рисунок 3.9 – Методологія управління проектами «Agile»

Agile спирається на чотири ключові цінності: люди й взаємодія важливіші за процеси й інструменти, працююче програмне забезпечення важливіше за вичерпну документацію, співпраця з замовником важливіша за контрактні переговори, реагування на зміни важливіше за слідування плану. До принципів належать регулярна доставка робочого продукту, тісна співпраця команди з бізнесом, прийняття змін навіть на пізніх етапах, підтримка сталого темпу роботи та постійне вдосконалення через ретроспективи.

Найпоширеніші фреймворки Agile – Scrum і Kanban, але існують і гібридні підходи. Scrum організовує роботу в ітерації (спринти), визначає ролі Product Owner, Scrum Master і команда розробки, а також церемонії: планування спринту, щоденний стендап, демонстрація результатів і ретроспектива. Kanban фокусується на візуалізації потоку робіт, обмеженні незавершених завдань і безперервному потоці доставки. До практик Agile належать інкрементальна розробка, тестування в циклі розробки, парне програмування, TDD, безперервна інтеграція і безперервне розгортання, автоматизація тестів і інфраструктури, а також часті релізи і збирання зворотного зв’язку від користувачів.

У Scrum Product Owner відповідає за пріоритизацію беклогу і максимізацію цінності, Scrum Master – за усунення перешкод і підтримку

процесу, команда – за доставку інкременту. Основні артефакти – Product Backlog, Sprint Backlog і Increment. Регулярні церемонії забезпечують синхронізацію: планування спринту визначає цілі і обсяг роботи, щоденний стендап координує дії команди, демо показує робочий інкремент зацікавленим сторонам, ретроспектива фокусується на покращенні процесу.

Впровадження починається з навчання команди і стейкхолдерів базовим принципам, вибору фреймворку і налаштування робочих артефактів. Рекомендується починати з малого пілота, встановити короткі ітерації, визначити чіткі критерії готовності (Definition of Done) і налаштувати інструменти для управління беклогом і автоматизації CI/CD. Важливо організувати регулярні зустрічі з бізнесом для пріоритизації і збирання зворотного зв'язку, а також проводити ретроспективи для поступового поліпшення процесів. Технічні практики – автоматичні тести, code review, інфраструктура як код – підвищують стабільність і швидкість доставки.

Agile дає швидше отримання цінності, кращу адаптацію до змін, підвищену прозорість і залученість замовника, а також покращену якість через часте тестування і інтеграцію. Ризики включають недостатню дисципліну в управлінні беклогом, хаотичну пріоритизацію, проблеми з масштабуванням у великих організаціях і можливі конфлікти ролей. Щоб мінімізувати ризики, потрібна чітка підтримка з боку менеджменту, інвестиції в автоматизацію, навчання команд і впровадження практик масштабування Agile при необхідності.

Для оцінки роботи в Agile використовують кількісні та якісні метрики: швидкість команди (velocity) для планування, lead time і cycle time для вимірювання часу від ідеї до релізу, частота релізів і час відновлення після інциденту для оцінки стабільності, а також коефіцієнт дефектів і задоволеність користувачів для контролю якості продукту. Метрики слід застосовувати обережно, фокусуючись на трендах і контексті, а не на ізольованих числах [38].

Agile особливо ефективний у проєктах з невизначеними або змінними вимогами, де важлива швидка перевірка гіпотез і часті релізи; для стабільних,

строго регламентованих середовищ може знадобитися адаптація підходів або поєднання з іншими методиками. Успіх Agile залежить від дисципліни в управлінні беклогом, підтримки з боку менеджменту, інвестицій в автоматизацію та культури постійного вдосконалення.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Архітектура програмного забезпечення

Архітектура програмного забезпечення – модульна, сервісно-орієнтована система, що поєднує інтерактивний фронтенд, API-бекенд, асинхронний обчислювальний шар для ML, надійне сховище даних і інфраструктуру для деплою та моніторингу. Система спроєктована для обробки великих наборів даних, інтерактивної візуалізації та повторюваного запуску ML-pipeline з можливістю масштабування.

React виконує роль основного фреймворку для побудови інтерфейсу користувача: він відповідає за структуру сторінок, управління станом, маршрутизацію, життєвий цикл компонентів і інтеграцію з API бекенду. React забезпечує модульність інтерфейсу (компоненти для авторизації, списку наборів, детального перегляду, конструктора візуалізацій) і дозволяє організувати повторно-використовувані UI-елементи та дизайн-систему з Figma.

D3 в цій платформі – це інструмент для точного перетворення даних у візуальні елементи і для реалізації складних інтерактивів і анімацій. React забезпечує каркас і управління станом, а D3 дає можливість реалізувати кастомні, високопродуктивні та інформативні візуалізації, необхідні для інтерактивного аналізу даних.

FastAPI в цьому проєкті – це інтерфейс, оркестратор і шлюз: він приймає запити, валідовує їх, координує асинхронні обчислення і повертає результати, тоді як PyTorch виконує фактичні ML-обчислення (тренування, інференс, explainability). Разом вони дають гнучку, продуктивну і масштабовану архітектуру для інтерактивної платформи візуалізації і аналізу даних.

PostgreSQL виступає як центральне реляційне сховище метаданих, конфігурацій, результатів аналізів і посилань на артефакти (файли, моделі). Воно забезпечує надійність, транзакційну цілісність, можливості складних

запитів і розширюваність (JSONB, геодані, індекси), що робить його основним джерелом істини для бекенду (FastAPI) і гарантує відтворюваність та версіонування результатів ML-pipeline.

В результаті отримуємо модульну веб-платформу з клієнтським шаром на React та D3 для інтерактивного інтерфейсу і візуалізацій, API-шаром на FastAPI для оркестрації запитів і аутентифікації, асинхронним обчислювальним шаром з PyTorch і воркерами (Celery/Redis) для виконання ML-pipeline, реляційним сховищем PostgreSQL для метаданих і налаштувань та S3-сумісним файловим сховищем для сирих даних і артефактів; компоненти взаємодіють через REST/WebSocket, черги задач і централізовані метрики, що забезпечує масштабованість, відтворюваність та інтерактивність.

Такий стек поєднує зручний, реактивний інтерфейс з потужним серверним ML-движком і надійним зберіганням даних, що дозволяє швидко проводити інтерактивний аналіз, масштабувати обробку великих наборів і забезпечувати прозорість та відтворюваність результатів.

4.2. Опис роботи платформи

Для опису роботи інтерактивної платформи для візуалізації масивів даних було вирішено побудувати діаграму Прецедентів (Рисунок 4.2).

Рисунок 4.2 – Діаграма прецедентів

ПРЕЦЕДЕНТ: АВТОРИЗАЦІЯ.

Ектор: Користувач.

Передумова: Користувач не авторизований в системі.

Післяумова: Користувач авторизований в системі.

Сценарій:

1. Користувач вводить свій пошту та пароль.
2. Користувач натискає кнопку «Увійти».
3. Користувач авторизований.
   3.1. Включення «Перевірка облікових даних».

ПРЕЦЕДЕНТ: КЕРУВАННЯ ДАТАСЕТАМИ

Ектор: Користувач.

Передумова: Користувач авторизований у системі.

Післяумова: Датасети оновлюються відповідно до виконаних дій: додавання або видалення.

Сценарій:

1. Користувач переходить до вкладки «Датасети».
2. На екрані з’являється список датасетів.
3. Користувач вибирає потрібний датасет.
4. Користувач видалив датасет.
   4.1. Включення «Додавання, перегляд та видалення статей»

ПРЕЦЕДЕНТ: КЕРУВАННЯ НАЛАШТУВАННЯМИ.

Ектор: Користувач.

Передумова: Користувач авторизований у системі.

Післяумова: Користувач оновив особисту інформацію про себе.

Сценарій:

1. Користувач переходить до вкладки «Налаштування».
2. Користувач переглянув можливості зміни профілю.
3. Користувач оновив особисту інформацію про себе.
   3.1. Включення «Зміна особистих даних»
   3.2. Включення «Додавання додаткової інформації про користувача»

ПРЕЦЕДЕНТ: ПЕРЕГЛЯД ДАТАСЕТУ

Ектор: Користувач.

Передумова: Користувач відкрив сторінку датасету

Післяумова: Користувач переглянув датасет.

Сценарій:

1. Користувач переходить до вкладки «Датасети».
2. На екрані з’являється список датасетів.
3. Користувач вибирає потрібний датасет.
4. Користувач переглянув датасет.

ПРЕЦЕДЕНТ: ПЕРЕГЛЯД КОНСТРУКТОРА ВІЗУАЛІЗАЦІЇ

Ектор: Користувач.

Передумова: Користувач відкрив сторінку датасету

Післяумова: Користувач переглянув конструктор візуалізації.

Сценарій:

1. Користувач переходить до вкладки «Датасети».
2. На екрані з’являється список датасетів.
3. Користувач вибирає потрібний датасет.
4. Користувач натискає «Запустити аналіз».
5. На екрані з’являється сторінка «Конструктор візуалізації».
6. Користувач переглянув методи візуалізації.

4.1. Розширення «Збереження візуалізації».

4.3. Інструкція з використання платформи

1. Початок роботи. Щоб мати можливість працювати з масивами даних, потрібно авторизуватися (Рисунок 4.2). Для цього переходимо на сторінку авторизації і вводимо логін та пароль. Після введення даних потрібно натиснути кнопку «Увійти» для отримання доступу до платформи. Додатково можна проставити галку «Запам’ятати мене», це допоможе уникнути повторної авторизації при заході на сайт протягом наступного тижня. Якщо під час авторизації користувач вводить невірні дані (пошта або пароль не відповідають формату), система виводить відповідне повідомлення (Рисунок 4.3). Також є можливість скинути пароль, для цього потрібно натиснути «Забули пароль?» після чого відбудеться редірект на сторінку з відновленням де потрібно буде вказати пошту, на яку було зареєстровано акаунт (Рисунок 4.4). За відсутності акаунту є можливість створити новий, натиснувши на кнопку «Зареєструватися», після чого відкриється нова форма для реєстрації, де потрібно вказати ім’я, пошту та пароль (Рисунок 4.5).

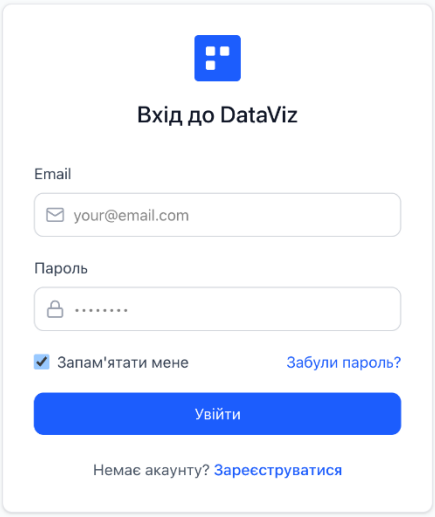


Рисунок 4.2 – Сторінка авторизації

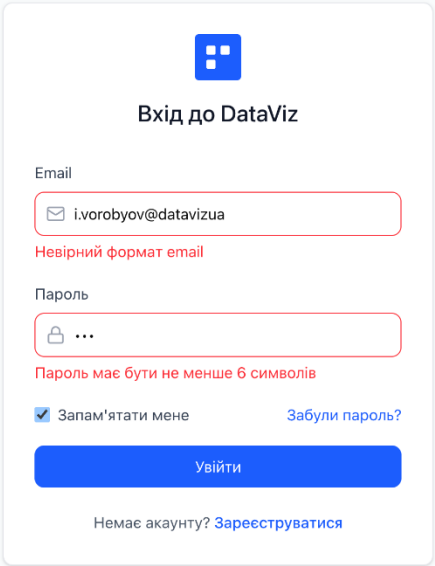


Рисунок 4.3 – Помилка авторизації

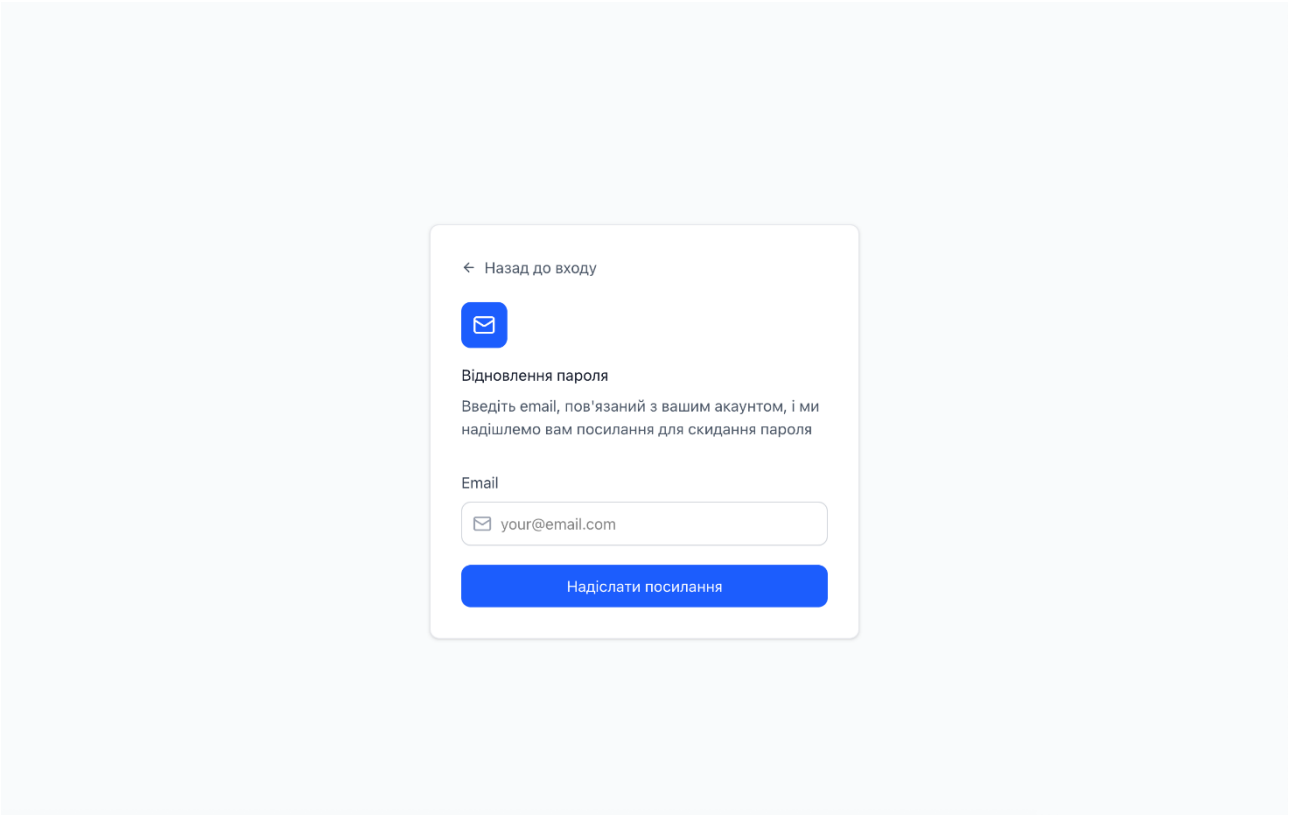


Рисунок 4.4 – Сторінка відновлення паролю

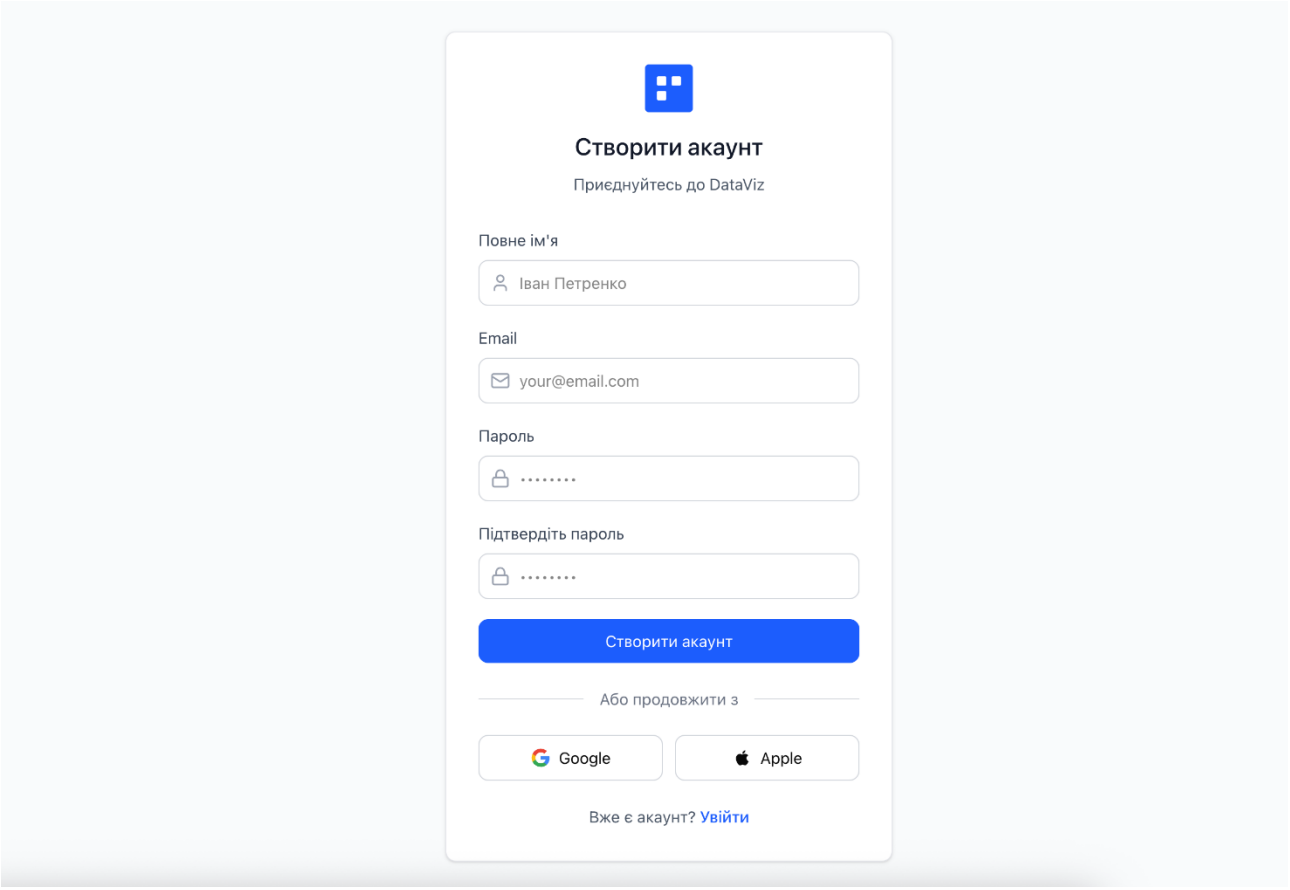


Рисунок 4.5 – Сторінка реєстрації

2. Після успішної авторизації користувач потрапляє на головну сторінку, де має змогу завантажити новий датасет, переглянути раніше завантажені, створити аналіз на основі вже завантажених датасетів а також бачить інформаю про останні датасети: назву, опис, інформацію про кількість рядків, стовпчиків та формат завантаженого датасету, прев'ю графіка або ж його статус (якщо він в процесі обробки) (Рисунок 4.6).

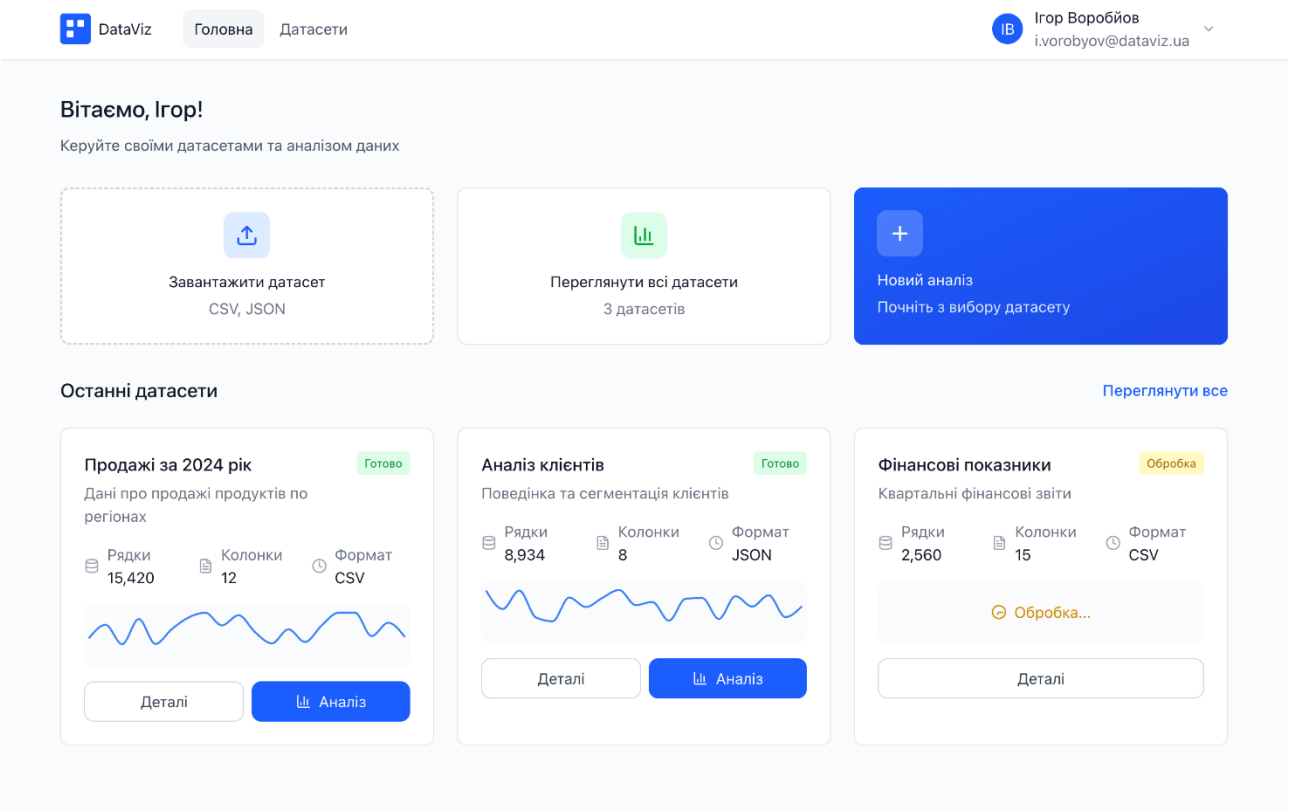


Рисунок 4.6 – Головна сторінка

3. Для перегляду списку раніше доданих датасетів потрібно натиснути кнопку «Датасети» вгорі сторінки або «Переглянути все» в групі «Останні датасети», після чого відкриється нова сторінка «Датасети» (Рисунок 4.7). На сторінці доступна можливість пошуку датасетів по назві та опису а також сортування. Доступні дії з датасетом на цій сторінці: завантажити новий (кнопка «Завантажити датасет»), відкрити існуючий (натиснути на його назву або рядок, або кнопку «Деталі»), переглянути його розмір, формат, статус та

дату останнього оновлення (новіші за замовчування знаходяться зверху), завантажити оригінальний датасет, видалити датасет.

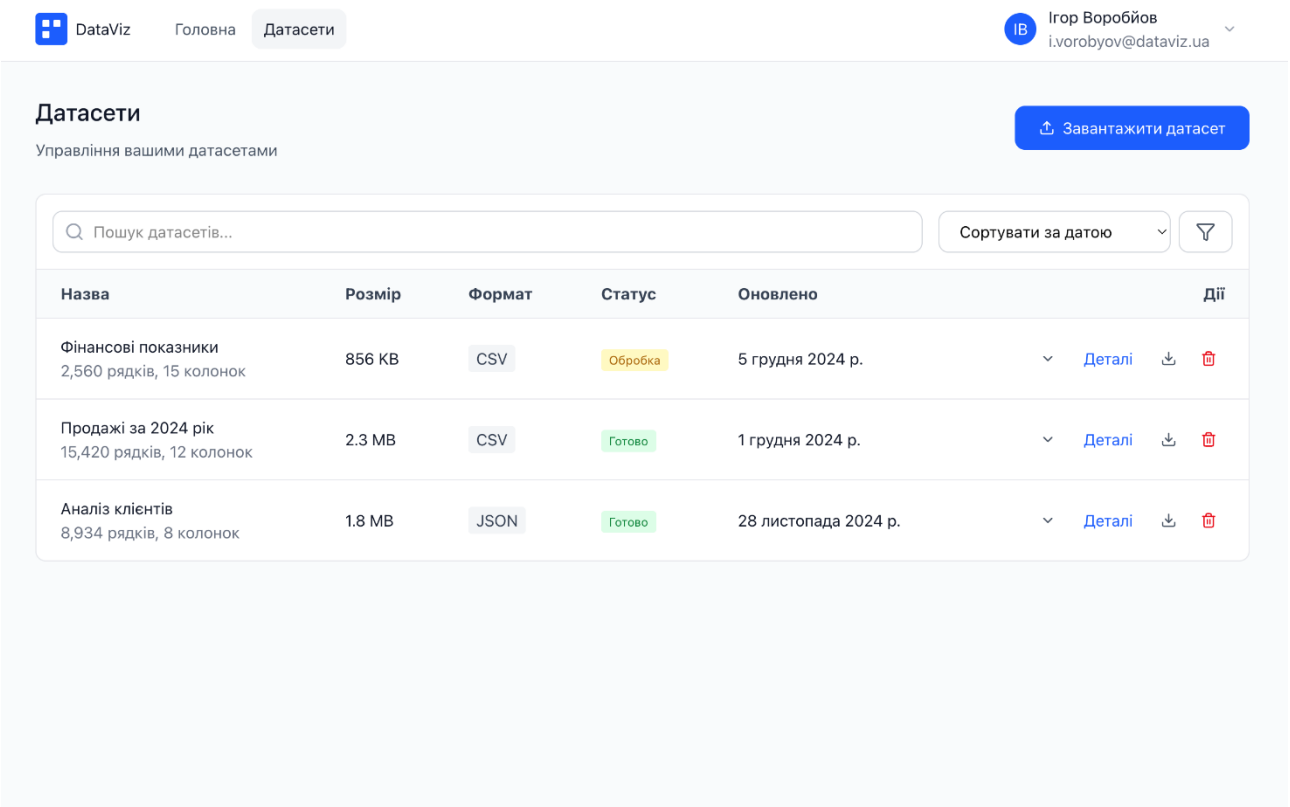


Рисунок 4.7 – Сторінка з датасетами

4. Для створення нового датасету потрібно натиснути кнопку «Завантажити датасет» на головній сторінці або на сторінці з датасетами. Після чого користувач потрапляє на сторінку, де потрібно вказати назву датасету (обов’язково), його опис, а також прикріпити сам файл перенос у відповідне вікно (Drag and drop) або натиснувши кнопку «Вибрати файл» та вибрати з локального сховища (Рисунок 4.8). Після заповнення всіх полів потрібно натиснути кнопку «Опублікувати» (Рисунок 4.14). Успішне додавання автоматично перенаправляє користувача на сторінку зі списком статей, де новостворена стаття відображатиметься в таблиці.

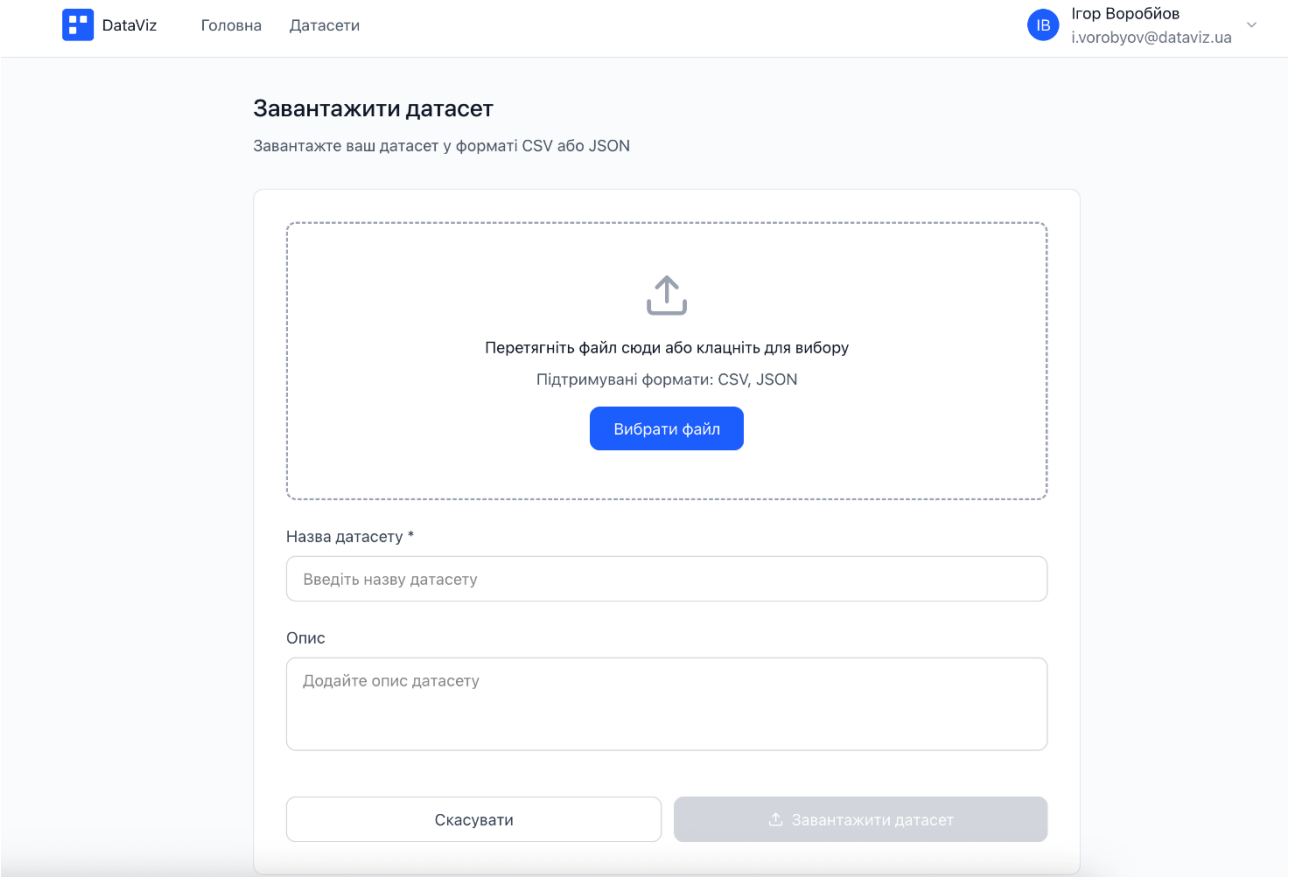


Рисунок 4.8 – Створення нового датасету

5. Для того, щоб переглянути створений датасет, потрібно натиснути кнопку «Деталі» на сторінці з датасетами або обрати серед запропонованих на головній сторінці в групі «Останні датасети». Після чого користувач потрапляє на сторінку, де може переглянути інформацію про датасет, а саме його назву, опис, кількість рядків, кількість колонок, розмір датасету, формат датасету. Також є можливість перегляду частини рядків з датасету і фільтрації даних. За допомогою кнопки «Експорт» можна завантажити оригінальний датасет, за допомогою кнопки «Запустити аналіз» – перейти на сторінку «Конструктор візуалізації» (Рисунок 4.10).

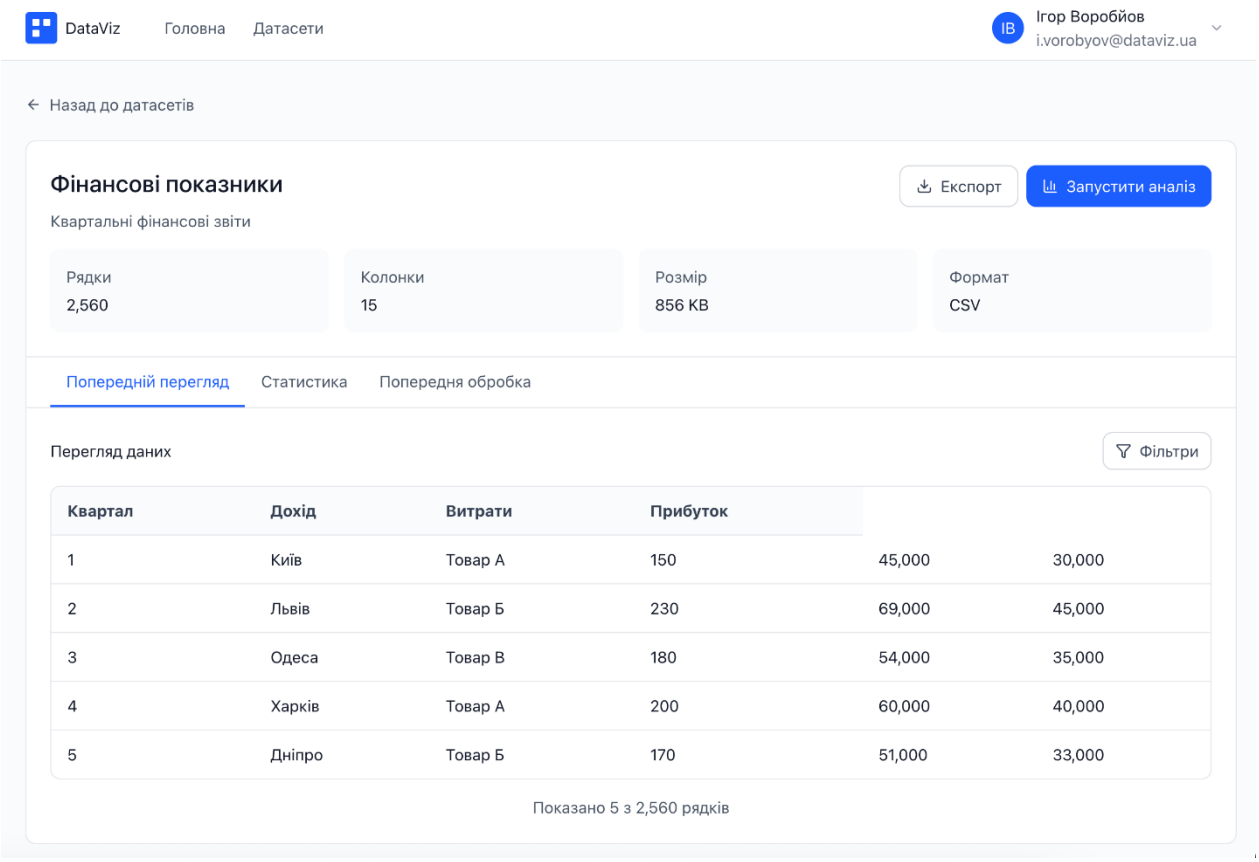


Рисунок 4.9 – Перегляд датасету

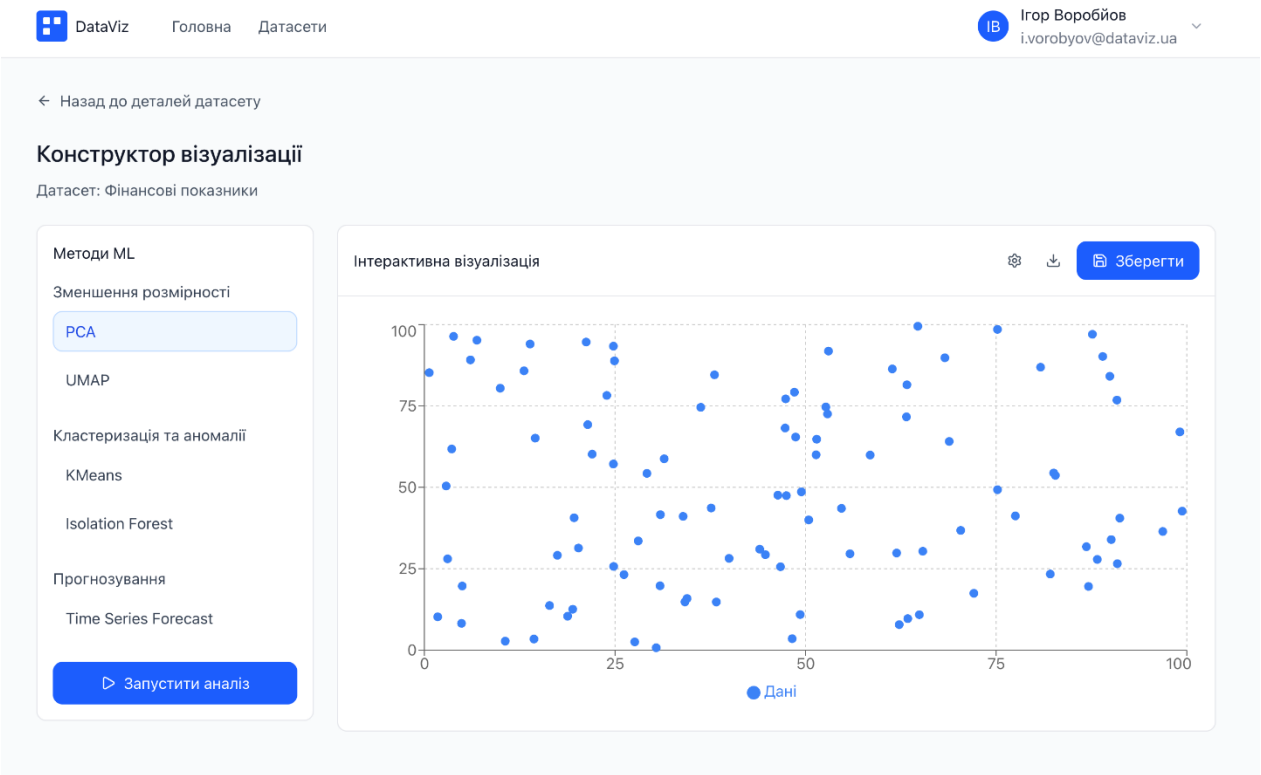


Рисунок 4.10 – Конструктор візуалізації

6. На сторінці «Конструктор візуалізації» доступні наступні методи: методи зменшення розмірності (PCA, UMAP), кластеризація та аномалії (KMeans, Isolation Forest), прогнозування (Time Series Forecast). На кластеризації можна вказати бажану кількість кластерів. На прогнозуванні можна обрати часову колонку, цільову змінну та горизонт прогнозу.

При кластеризації буде вказано середні значення та кількість кластерів, а також відображено на діаграмі (Рисунок 4.11). Також є можливість вивантажити результат кластеризації в новий датасет у форматі CSV або JSON, або ж зберегти графік у форматі SVG.

При виборі методу прогнозування також буде доступне вивантаження даних (у форматі CSV або JSON) або графіку, а також буде вказано прогноз на вказаний період (мінімальне та максимальне прогнозоване значення), точність моделі у відсотках, а також тренд (зростання, спадання або бічний рух) (Рисунок 4.12).

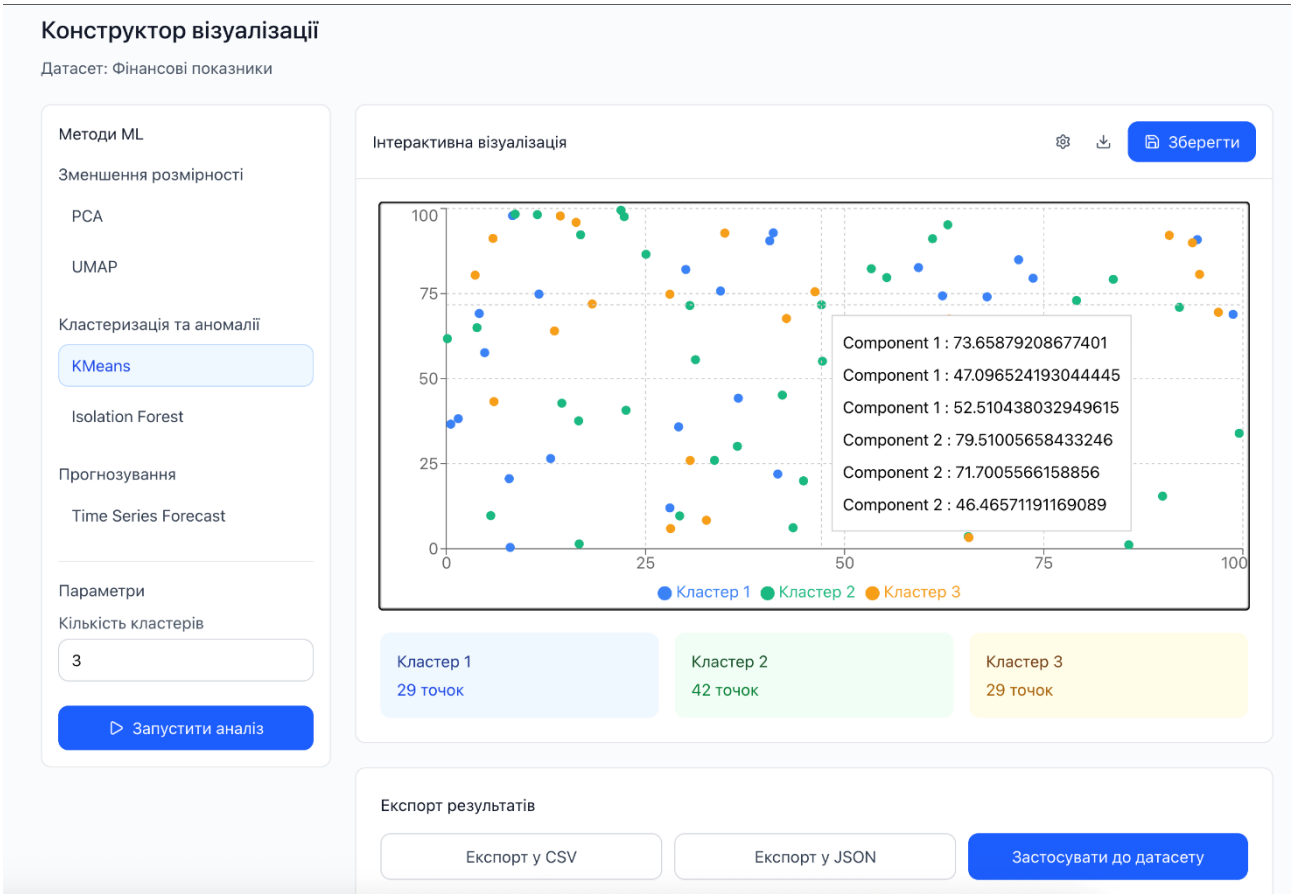


Рисунок 4.11 – Приклад кластеризації

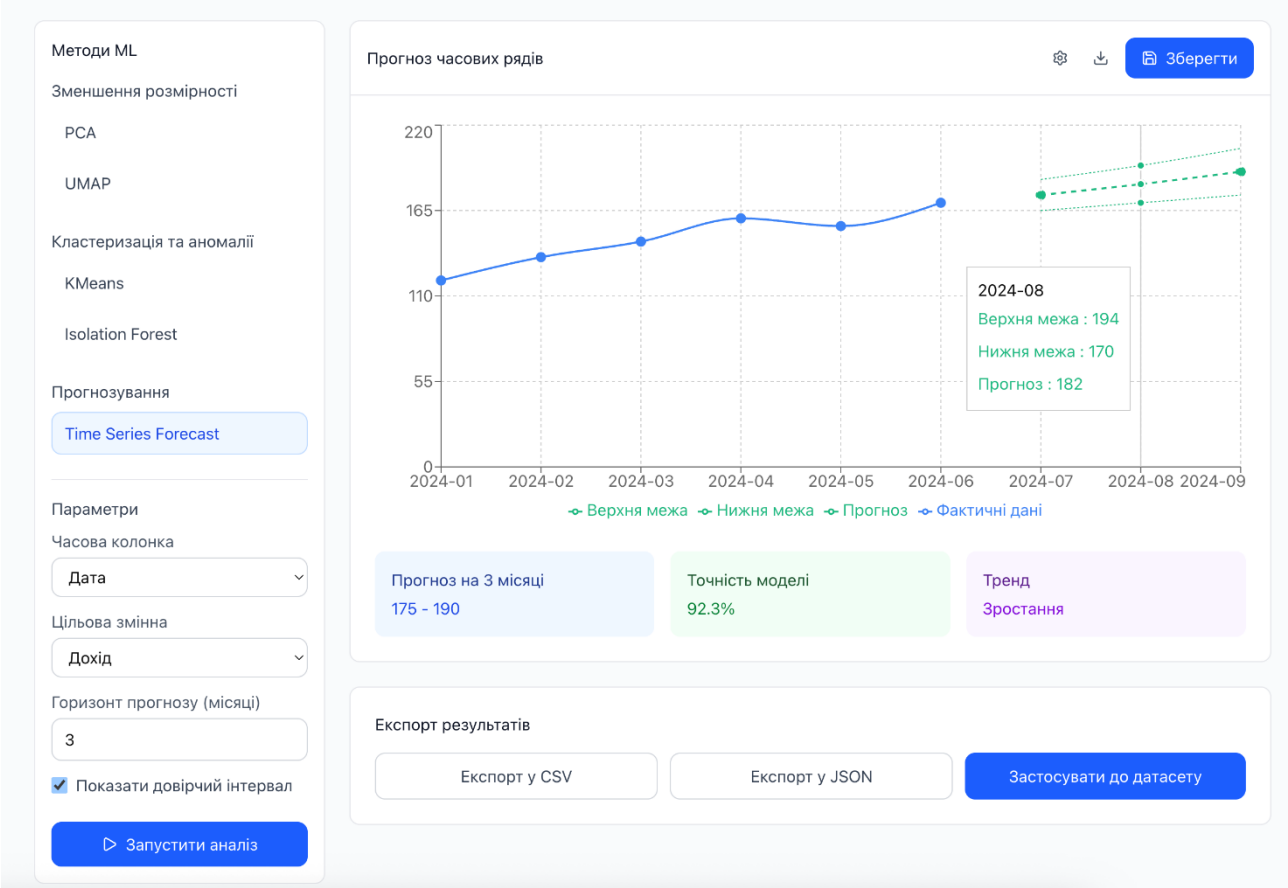


Рисунок 4.12 – Приклад прогнозування

7. Для налаштування профілю необхідно вибрати вкладку «Налаштування» правому верхньому кутку, після чого відкриється сторінка «Профіль користувача» (Рисунок 4.13). На сторінці в групі «Особиста інформація» користувач може змінити відповідні поля: ім’я, пошту, компанію, місту та країну. За потреби є можливість змінити пароль в групі «Безпека акаунту», а також керувати двофакторною автентифікацією (ввімкнути або вимкнути).

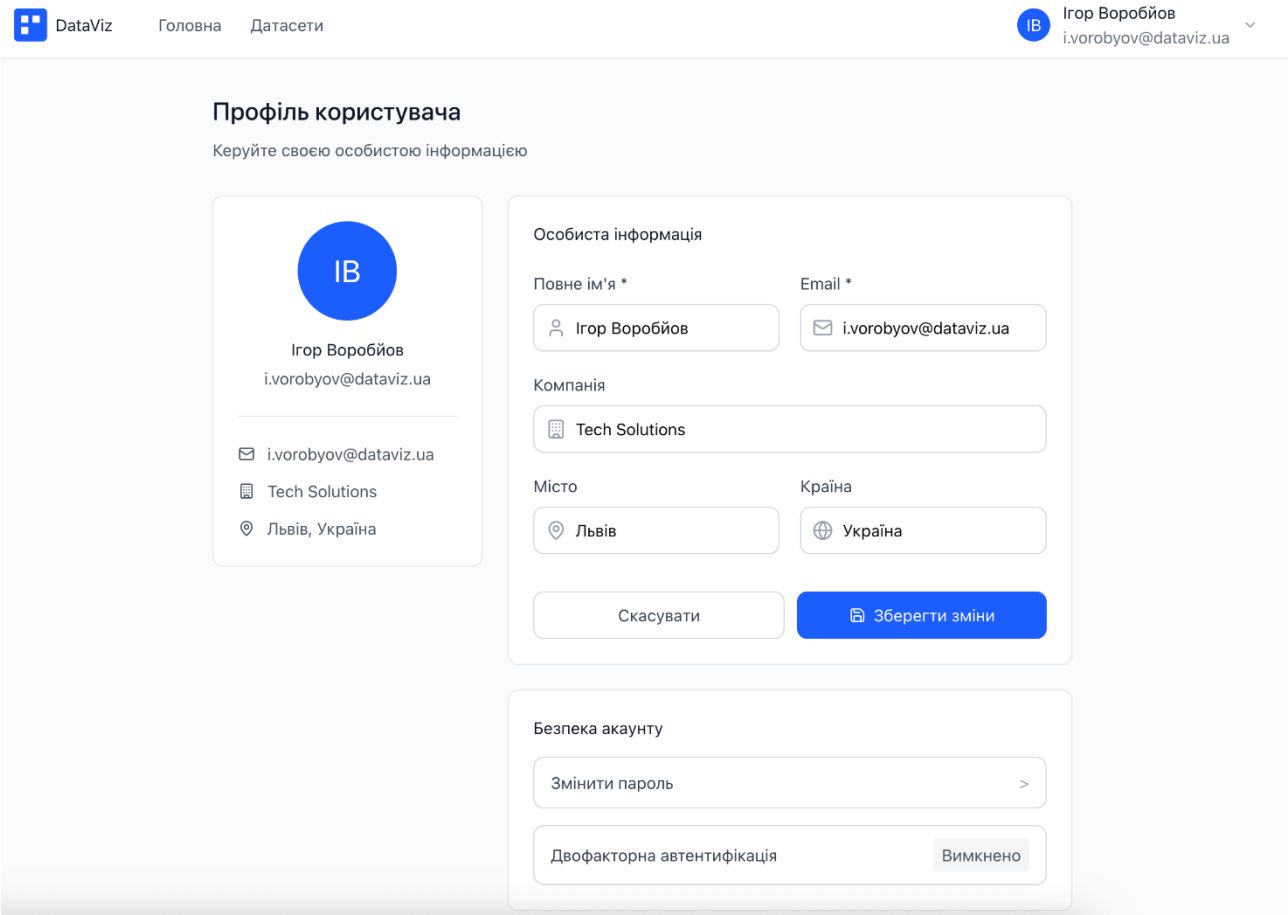


Рисунок 4.13 – Профіль користувача

ВИСНОВКИ

Розробка інтерактивної платформи для візуалізації масивів даних із застосуванням сучасних методів машинного навчання є актуальною задачею для аналітики, наукових досліджень та освітнього процесу. У межах кваліфікаційної роботи було поставлено завдання створити інструмент, що поєднує зручний інтерфейс для дослідника, надійний бекенд для оркестрації обчислень і модульний ML-шар для виконання pipeline-аналізів. Робота пройшла апробацію, результати можуть бути використані в практичних дослідженнях і навчальних курсах.

Основні результати роботи:

- визначено актуальність теми та обґрунтовано потребу в інтерактивних інструментах для аналізу великих даних;
- сформульовано основні вимоги до платформи;
- проведено огляд існуючих рішень, виявлено їхні переваги та обмеження;
- описано архітектурні рішення та вибір технологічного стеку (React в поєднанні з D3, FastAPI, PyTorch, PostgreSQL);
- обґрунтовано підхід до організації ML pipeline, включно з препроцесингом, зниженням розмірності, кластеризацією, виявленням аномалій і прогнозуванням;
- реалізовано прототип інтерфейсу з інтерактивними візуалізаціями;
- розроблено бекенд сервіси для управління наборами даних, запуску асинхронних задач і надання результатів через REST;
- реалізовано механізми збереження налаштувань аналізів, версіонування наборів і артефактів моделей для відтворюваності;
- підготовлено інструкцію користувача та рекомендації з експлуатації.

За результатами роботи над дипломних проектом було опубліковано тези на міжнародній науковій конференції.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Ultimate Tableau Review: Know What Users Really Think in 2025 [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://upsolve.ai/blog/tableau-review/ – Назва з екрану.
2. A Detailed Guide to Tableau Architecture: Desktop and Server [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.datacamp.com/tutorial/exploring-tableau-architecture-desktop-and-server/ – Назва з екрану.
3. Ultimate Tableau Review: Know What Users Really Think in 2025 [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://upsolve.ai/blog/tableau-review/ – Назва з екрану.
4. Looker business intelligence platform embedded analytics | Google Cloud [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://cloud.google.com/looker/ – Назва з екрану.
5. The Modern Data Platform and LookML | Google Skills [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.skills.google/course_templates/226/ – Назва з екрану.
6. Looker Review 2026: Pricing, Features, Pros & Cons, Ratings & More [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://research.com/software/reviews/looker/ – Назва з екрану.
7. Technical Documentation | Grafana Labs [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://grafana.com/docs/ – Назва з екрану.
8. Red Hat – Grafana overview [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.redhat.com/en/topics/analytics/what-is-grafana/ – Назва з екрану.
9. Grafana Labs Reviews, Ratings & Features 2025 | Gartner Peer Insights [Електронний ресурс]. – Режим доступу до ресурсу: URL:

https://www.gartner.com/reviews/market/infrastructure-monitoring-tools/vendor/grafana-labs – Назва з екрану.

10. Kibana: Visualize, explore and manage data in Elasticsearch | Elastic [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.elastic.co/kibana – Назва з екрану.
11. The Elastic Stack | Elastic Docs [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.elastic.co/docs/get-started/the-stack – Назва з екрану.
12. Elastic Stack Reviews 2025: Details, Pricing & Features | G2 [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.g2.com/products/elastic-stack/reviews – Назва з екрану.
13. What is Figma? – Figma Learn – Help [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma – Назва з екрану.
14. Core Features of the Figma Platform [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.figma.com/legal/core-features/ – Назва з екрану.
15. Figma UX Design Tools Review for 2025 [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://cpoclub.com/tools/figma-review/ – Назва з екрану.
16. Overview of React.js [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.patterns.dev/react/ – Назва з екрану.
17. React introduction [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.geeksforgeeks.org/reactjs/reactjs-introduction/ – Назва з екрану.
18. A brief review of React.js framework, for those wondering whether to use it or not [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://antsstyle.medium.com/a-brief-review-of-the-react-js-framework-for-those-wondering-whether-to-use-it-or-not-e85d21b68158 – Назва з екрану.
19. What is D3? | D3 by Observable [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://d3js.org/what-is-d3 – Назва з екрану.

20. Introduction to D3.js [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.d3indepth.com/introduction/ – Назва з екрану.
21. 10 Years of Open-Source Visualization: Did I learn anything from D3.js? [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://news.ycombinator.com/item?id=26240339 – Назва з екрану.
22. An Introduction to Using FastAPI | Refine [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://refine.dev/blog/introduction-to-fast-api/ – Назва з екрану.
23. A Close Look at a FastAPI Example Application = Real Python [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://realpython.com/fastapi-python-web-apis/ – Назва з екрану.
24. FastAPI – The Good, the bad and the ugly. – DEV Community [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://dev.to/fuadrafid/fastapi-the-good-the-bad-and-the-ugly-20ob – Назва з екрану.
25. PostgreSQL: About [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.postgresql.org/about/ – Назва з екрану.
26. What is PostgreSQL? How It Works, Use Cases and Resources | DataCamp [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.datacamp.com/blog/what-is-postgresql-introduction – Назва з екрану.
27. PostgreSQL Review | Data System Reviews [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://datasystemreviews.com/postgresql-review.html – Назва з екрану.
28. What is PyTorch? | IBM [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.ibm.com/think/topics/pytorch – Назва з екрану.
29. Learning PyTorch with Examples – PyTorch Tutorials [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://docs.pytorch.org/tutorials/beginner/pytorch_with_examples.html – Назва з екрану.

30. A comprehensive review of PyTorch | by Henilsinh Raj | Medium [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://medium.com/@henilsinhrajraj/a-comprehensive-review-of-pytorch-c28ce6f4ada3 – Назва з екрану.
31. An overview of Visual Studio Code for front-end developers | by Vinh Le [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://medium.com/free-code-camp/an-overview-of-visual-studio-code-for-front-end-developers-49a4aa0771fb – Назва з екрану.
32. What is Visual Studio Code? [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.hostinger.com/uk/tutorials/what-is-vs-code – Назва з екрану.
33. Visual Studio Code Review – General – The freeCodeCamp Forum [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://forum.freecodecamp.org/t/visual-studio-code-review/56993 – Назва з екрану.
34. Git – What is Git? [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F – Назва з екрану.
35. How to Use Git? Tutorials, Workflows & Commands | Atlassian [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.atlassian.com/git – Назва з екрану.
36. Git Reviews 2025: Details, Pricing & Features | G2 [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.g2.com/products/git/reviews – Назва з екрану.
37. What Is Agile? And When to Use It [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.coursera.org/articles/what-is-agile-a-beginners-guide – Назва з екрану.
38. What is Agile? [Електронний ресурс]. – Режим доступу до ресурсу: URL: https://www.atlassian.com/agile – Назва з екрану.

ДОДАТОК А. ВИХІДНІ КОДИ

СТОРІНКА АВТОРИЗАЦІЇ

```
import { useState } from 'react';
import { Mail, Lock } from 'lucide-react';

interface LoginPageProps {
  onLogin: (email: string, password: string) => void;
  onNavigateToRegister: () => void;
  onNavigateToForgot: () => void;
}

export function LoginPage({ onLogin, onNavigateToRegister, onNavigateToForgot }:
LoginPageProps) {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [errors, setErrors] = useState({ email: '', password: '' });

  const validateForm = () => {
    const newErrors = { email: '', password: '' };
    let isValid = true;

    if (!email) {
      newErrors.email = 'Email обов\'язковий';
      isValid = false;
    } else if (!/\S+@\S+\.\S+/.test(email)) {
      newErrors.email = 'Невірний формат email';
      isValid = false;
    }

    if (!password) {
      newErrors.password = 'Пароль обов\'язковий';
      isValid = false;
    } else if (password.length < 6) {
      newErrors.password = 'Пароль має бути не менше 6 символів';
      isValid = false;
    }

    setErrors(newErrors);
    return isValid;
  };

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    if (validateForm()) {
      fetch('http://localhost:3001/api/signin', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
```

```
      },
      body: JSON.stringify({ email, password }),
    })
    .then(response => {
      if (response.ok) {
        onLogin(email, password);
      } else {
        // Handle error response
        setErrors({ ...errors, email: 'Login failed', password: '' });
      }
    })
    .catch(error => {
      // Handle network error
      setErrors({ ...errors, email: 'Network error', password: '' });
    });
  }
 };

 return (
  <div className="min-h-screen bg-gray-50 flex items-center justify-center p-4">
   <div className="w-full max-w-md">
    <div className="bg-white rounded-lg shadow-sm border border-gray-200 p-8">
     <div className="text-center mb-8">
      <div className="w-12 h-12 bg-blue-600 rounded mx-auto mb-4 flex items-center
justify-center">
       <svg width="24" height="24" viewBox="0 0 18 18" fill="none"
xmlns="http://www.w3.org/2000/svg">
        <rect x="0" y="0" width="7" height="7" rx="1" fill="white" />
        <rect x="11" y="0" width="7" height="7" rx="1" fill="white" />
        <rect x="0" y="11" width="7" height="7" rx="1" fill="white" />
       </svg>
      </div>
      <h1 className="text-gray-900 mb-2">Вхід до DataViz</h1>
     </div>

     <form onSubmit={handleSubmit} className="space-y-5">
      <div>
       <label htmlFor="email" className="block text-gray-700 mb-2">
        Email
       </label>
       <div className="relative">
        <Mail className="absolute left-3 top-1/2 -translate-y-1/2 w-5 h-5 text-gray-400"
/>
        <input
         id="email"
         type="email"
         value={email}
         onChange={(e) => setEmail(e.target.value)}
         className={`w-full pl-10 pr-4 py-2.5 border ${errors.email ? 'border-red-500' :
'border-gray-300'} rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500
focus:border-transparent`}
         placeholder="your@email.com"
```

```
            />
          </div>
          {errors.email && <p className="text-red-500 mt-1">{errors.email}</p>}
        </div>

        <div>
          <label htmlFor="password" className="block text-gray-700 mb-2">
            Пароль
          </label>
          <div className="relative">
            <Lock className="absolute left-3 top-1/2 -translate-y-1/2 w-5 h-5 text-gray-400"
/>
            <input
              id="password"
              type="password"
              value={password}
              onChange={(e) => setPassword(e.target.value)}
              className={`w-full pl-10 pr-4 py-2.5 border ${errors.password ? 'border-red-
500' : 'border-gray-300'} rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500
focus:border-transparent`}
              placeholder="••••••••"
            />
          </div>
          {errors.password && <p className="text-red-500 mt-1">{errors.password}</p>}
        </div>

        <div className="flex items-center justify-between">
          <label className="flex items-center">
            <input type="checkbox" className="w-4 h-4 text-blue-600 border-gray-300
rounded focus:ring-blue-500" />
            <span className="ml-2 text-gray-700">Запам'ятати мене</span>
          </label>
          <button
            type="button"
            onClick={onNavigateToForgot}
            className="text-blue-600 hover:text-blue-700"
          >
            Забули пароль?
          </button>
        </div>

        <button
          type="submit"
          className="w-full bg-blue-600 text-white py-2.5 rounded-lg hover:bg-blue-700
transition-colors"
        >
          Увійти
        </button>
      </form>

      <div className="mt-6 text-center">
        <p className="text-gray-600">
```

```
                Немає акаунту?{' '}
                <button
                  onClick={onNavigateToRegister}
                  className="text-blue-600 hover:text-blue-700"
                >
                  Зареєструватися
                </button>
              </p>
            </div>
          </div>
        </div>
      </div>
    );
}
```

СТОРІНКА РЕЄСТРАЦІЇ

```
import { useState } from 'react';
import { Mail, Lock, User } from 'lucide-react';

interface RegisterPageProps {
  onRegister: (name: string, email: string, password: string) => void;
  onNavigateToLogin: () => void;
}

export function RegisterPage({ onRegister, onNavigateToLogin }: RegisterPageProps) {
  const [formData, setFormData] = useState({
    name: '',
    email: '',
    password: '',
    confirmPassword: ''
  });
  const [errors, setErrors] = useState({
    name: '',
    email: '',
    password: '',
    confirmPassword: ''
  });

  const validateForm = () => {
    const newErrors = {
      name: '',
      email: '',
      password: '',
      confirmPassword: ''
    };
    let isValid = true;

    if (!formData.name) {
      newErrors.name = 'Ім\'я обов\'язкове';
      isValid = false;
    }
```

```
  if (!formData.email) {
    newErrors.email = 'Email обов\'язковий';
    isValid = false;
  } else if (!/\S+@\S+\.\S+/.test(formData.email)) {
    newErrors.email = 'Невірний формат email';
    isValid = false;
  }

  if (!formData.password) {
    newErrors.password = 'Пароль обов\'язковий';
    isValid = false;
  } else if (formData.password.length < 6) {
    newErrors.password = 'Пароль має бути не менше 6 символів';
    isValid = false;
  }

  if (formData.password !== formData.confirmPassword) {
    newErrors.confirmPassword = 'Паролі не співпадають';
    isValid = false;
  }

  setErrors(newErrors);
  return isValid;
};

const handleSubmit = (e: React.FormEvent) => {
  e.preventDefault();
  if (validateForm()) {
    fetch('http://localhost:3001/api/register', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        name: formData.name,
        email: formData.email,
        password: formData.password,
      }),
    })
    .then(response => {
      if (!response.ok) {
        throw new Error('Network response was not ok');
      }
      return response.json();
    })
    .then(data => {
      // Handle successful registration (e.g., navigate to login)
      console.log('Registration successful:', data);
      onNavigateToLogin();
    })
    .catch(error => {
      console.error('There was a problem with the registration:', error);
```

```
    });
  }
 };

 const handleChange = (field: string, value: string) => {
  setFormData({ ...formData, [field]: value });
 };

 return (
  <div className="min-h-screen bg-gray-50 flex items-center justify-center p-4">
   <div className="w-full max-w-md">
    <div className="bg-white rounded-lg shadow-sm border border-gray-200 p-8">
     <div className="text-center mb-8">
      <div className="w-12 h-12 bg-blue-600 rounded mx-auto mb-4 flex items-center justify-center">
       <svg width="24" height="24" viewBox="0 0 18 18" fill="none" xmlns="http://www.w3.org/2000/svg">
        <rect x="0" y="0" width="7" height="7" rx="1" fill="white" />
        <rect x="11" y="0" width="7" height="7" rx="1" fill="white" />
        <rect x="0" y="11" width="7" height="7" rx="1" fill="white" />
       </svg>
      </div>
      <h1 className="text-gray-900 mb-2">Створити акаунт</h1>
      <p className="text-gray-600">Приєднуйтесь до DataViz</p>
     </div>

     <form onSubmit={handleSubmit} className="space-y-5">
      <div>
       <label htmlFor="name" className="block text-gray-700 mb-2">
        Повне ім'я
       </label>
       <div className="relative">
        <User className="absolute left-3 top-1/2 -translate-y-1/2 w-5 h-5 text-gray-400" />
        <input
         id="name"
         type="text"
         value={formData.name}
         onChange={(e) => handleChange('name', e.target.value)}
         className={`w-full pl-10 pr-4 py-2.5 border ${errors.name ? 'border-red-500' : 'border-gray-300'} rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent`}
         placeholder="Іван Петренко"
        />
       </div>
       {errors.name && <p className="text-red-500 mt-1">{errors.name}</p>}
      </div>

      <div>
       <label htmlFor="email" className="block text-gray-700 mb-2">
        Email
       </label>
```

```
            <div className="relative">
              <Mail className="absolute left-3 top-1/2 -translate-y-1/2 w-5 h-5 text-gray-400"
/>
              <input
                id="email"
                type="email"
                value={formData.email}
                onChange={(e) => handleChange('email', e.target.value)}
                className={`w-full pl-10 pr-4 py-2.5 border ${errors.email ? 'border-red-500' :
'border-gray-300'} rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500
focus:border-transparent`}
                placeholder="your@email.com"
              />
            </div>
            {errors.email && <p className="text-red-500 mt-1">{errors.email}</p>}
          </div>

          <div>
            <label htmlFor="password" className="block text-gray-700 mb-2">
              Пароль
            </label>
            <div className="relative">
              <Lock className="absolute left-3 top-1/2 -translate-y-1/2 w-5 h-5 text-gray-400"
/>
              <input
                id="password"
                type="password"
                value={formData.password}
                onChange={(e) => handleChange('password', e.target.value)}
                className={`w-full pl-10 pr-4 py-2.5 border ${errors.password ? 'border-red-
500' : 'border-gray-300'} rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500
focus:border-transparent`}
                placeholder="••••••••"
              />
            </div>
            {errors.password && <p className="text-red-500 mt-1">{errors.password}</p>}
          </div>

          <div>
            <label htmlFor="confirmPassword" className="block text-gray-700 mb-2">
              Підтвердіть пароль
            </label>
            <div className="relative">
              <Lock className="absolute left-3 top-1/2 -translate-y-1/2 w-5 h-5 text-gray-400"
/>
              <input
                id="confirmPassword"
                type="password"
                value={formData.confirmPassword}
                onChange={(e) => handleChange('confirmPassword', e.target.value)}
```

```
            className={`w-full pl-10 pr-4 py-2.5 border ${errors.confirmPassword ? 'border-red-500' : 'border-gray-300'} rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent`}
            placeholder="••••••••"
           />
          </div>
          {errors.confirmPassword && <p className="text-red-500 mt-1">{errors.confirmPassword}</p>}
        </div>

        <button
         type="submit"
         className="w-full bg-blue-600 text-white py-2.5 rounded-lg hover:bg-blue-700 transition-colors"
        >
         Створити акаунт
        </button>
       </form>

       <div className="mt-6">
        <div className="relative">
         <div className="absolute inset-0 flex items-center">
          <div className="w-full border-t border-gray-300"></div>
         </div>
         <div className="relative flex justify-center">
          <span className="px-4 bg-white text-gray-500">Або продовжити з</span>
         </div>
        </div>

        <div className="mt-6 grid grid-cols-2 gap-3">
         <button className="flex items-center justify-center gap-2 px-4 py-2.5 border border-gray-300 rounded-lg hover:bg-gray-50 transition-colors">
          <svg className="w-5 h-5" viewBox="0 0 24 24">
           <path fill="#4285F4" d="M22.56 12.25c0-.78-.07-1.53-.2-2.25H12v4.26h5.92c-.26 1.37-1.04 2.53-2.21 3.31v2.77h3.57c2.08-1.92 3.28-4.74 3.28-8.09z"/>
           <path fill="#34A853" d="M12 23c2.97 0 5.46-.98 7.28-2.66l-3.57-2.77c-.98.66-2.23 1.06-3.71 1.06-2.86 0-5.29-1.93-6.16-4.53H2.18v2.84C3.99 20.53 7.7 23 12 23z"/>
           <path fill="#FBBC05" d="M5.84 14.09c-.22-.66-.35-1.36-.35-2.09s.13-1.43.35-2.09V7.07H2.18C1.43 8.55 1 10.22 1 12s.43 3.45 1.18 4.93l2.85-2.22.81-.62z"/>
           <path fill="#EA4335" d="M12 5.38c1.62 0 3.06.56 4.21 1.64l3.15-3.15C17.45 2.09 14.97 1 12 1 7.7 1 3.99 3.47 2.18 7.07l3.66 2.84c.87-2.6 3.3-4.53 6.16-4.53z"/>
          </svg>
          Google
         </button>
         <button className="flex items-center justify-center gap-2 px-4 py-2.5 border border-gray-300 rounded-lg hover:bg-gray-50 transition-colors">
          <svg className="w-5 h-5" viewBox="0 0 24 24" fill="currentColor">
           <path d="M17.05 20.28c-.98.95-2.05.8-3.08.35-1.09-.46-2.09-.48-3.24 0-1.44.62-2.2.44-3.06-.35C2.79 15.25 3.51 7.59 9.05 7.31c1.35.07 2.29.74 3.08.8 1.18-.24 2.31-.93 3.57-.84 1.51.12 2.65.72 3.4 1.8-3.12 1.87-2.38 5.98.48 7.13-.57 1.5-1.31 2.99-2.54 4.09l.01-.01zM12.03 7.25c-.15-2.23 1.66-4.07 3.74-4.25.29 2.58-2.34 4.5-3.74 4.25z"/>
          </svg>
```

```
                Apple
              </button>
            </div>
          </div>

          <div className="mt-6 text-center">
            <p className="text-gray-600">
              Вже є акаунт?{' '}
              <button
                onClick={onNavigateToLogin}
                className="text-blue-600 hover:text-blue-700"
              >
                Увійти
              </button>
            </p>
          </div>
        </div>
      </div>
    );
}
```

СТОРІНКА ВІДНОВЛЕННЯ ПАРОЛЮ

```
import { useState } from 'react';
import { Mail, ArrowLeft, CheckCircle } from 'lucide-react';

interface ForgotPasswordPageProps {
  onNavigateToLogin: () => void;
}

export function ForgotPasswordPage({ onNavigateToLogin }: ForgotPasswordPageProps) {
  const [email, setEmail] = useState('');
  const [error, setError] = useState('');
  const [isSubmitted, setIsSubmitted] = useState(false);

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();

    if (!email) {
      setError('Email обов\'язковий');
      return;
    }

    if (!/\S+@\S+\.\S+/.test(email)) {
      setError('Невірний формат email');
      return;
    }

    setError('');
    fetch('http://localhost:3001/api/reset-password', {
      method: 'POST',
      headers: {
      'Content-Type': 'application/json',
```

```
    },
    body: JSON.stringify({ email }),
   })
    .then(response => {
    if (!response.ok) {
     throw new Error('Network response was not ok');
    }
    return response.json();
    })
    .then(() => {
    setIsSubmitted(true);
    })
    .catch(error => {
    setError('Сталася помилка при надсиланні запиту');
    });
 };

 if (isSubmitted) {
  return (
   <div className="min-h-screen bg-gray-50 flex items-center justify-center p-4">
    <div className="w-full max-w-md">
     <div className="bg-white rounded-lg shadow-sm border border-gray-200 p-8">
      <div className="text-center">
       <div className="w-16 h-16 bg-green-100 rounded-full mx-auto mb-4 flex items-
center justify-center">
        <CheckCircle className="w-8 h-8 text-green-600" />
       </div>
       <h1 className="text-gray-900 mb-2">Перевірте свій email</h1>
       <p className="text-gray-600 mb-6">
        Ми надіслали посилання для скидання пароля на адресу {email}
       </p>
       <button
        onClick={onNavigateToLogin}
        className="w-full bg-blue-600 text-white py-2.5 rounded-lg hover:bg-blue-700
transition-colors"
       >
        Повернутися до входу
       </button>
      </div>
     </div>
    </div>
   </div>
  );
 }

 return (
  <div className="min-h-screen bg-gray-50 flex items-center justify-center p-4">
   <div className="w-full max-w-md">
    <div className="bg-white rounded-lg shadow-sm border border-gray-200 p-8">
     <button
      onClick={onNavigateToLogin}
      className="flex items-center gap-2 text-gray-600 hover:text-gray-900 mb-6"
```

```
            >
              <ArrowLeft className="w-4 h-4" />
              Назад до входу
            </button>

            <div className="mb-8">
              <div className="w-12 h-12 bg-blue-600 rounded-lg mb-4 flex items-center justify-center">
                <Mail className="w-6 h-6 text-white" />
              </div>
              <h1 className="text-gray-900 mb-2">Відновлення пароля</h1>
              <p className="text-gray-600">
                Введіть email, пов'язаний з вашим акаунтом, і ми надішлемо вам посилання для скидання пароля
              </p>
            </div>

            <form onSubmit={handleSubmit} className="space-y-5">
              <div>
                <label htmlFor="email" className="block text-gray-700 mb-2">
                  Email
                </label>
                <div className="relative">
                  <Mail className="absolute left-3 top-1/2 -translate-y-1/2 w-5 h-5 text-gray-400" />
                  <input
                    id="email"
                    type="email"
                    value={email}
                    onChange={(e) => setEmail(e.target.value)}
                    className={`w-full pl-10 pr-4 py-2.5 border ${error ? 'border-red-500' : 'border-gray-300'} rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent`}
                    placeholder="your@email.com"
                  />
                </div>
                {error && <p className="text-red-500 mt-1">{error}</p>}
              </div>

              <button
                type="submit"
                className="w-full bg-blue-600 text-white py-2.5 rounded-lg hover:bg-blue-700 transition-colors"
              >
                Надіслати посилання
              </button>
            </form>
          </div>
        </div>
      </div>
    );
}
```

ФОРМА НАВІГАЦІЇ

```
import { useState } from 'react';
import { ChevronDown, User, Settings, LogOut } from 'lucide-react';
import type { User as UserType, Page } from '../../App';

interface NavbarProps {
  user: UserType;
  onLogout: () => void;
  onNavigate: (page: Page) => void;
  currentPage?: Page;
}

export function Navbar({ user, onLogout, onNavigate, currentPage }: NavbarProps) {
  const [isDropdownOpen, setIsDropdownOpen] = useState(false);

  const getInitials = (name: string) => {
    return name
      .split(' ')
      .map(n => n[0])
      .join('')
      .toUpperCase()
      .slice(0, 2);
  };

  return (
    <nav className="bg-white border-b border-gray-200">
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
        <div className="flex justify-between items-center h-16">
          <div className="flex items-center gap-8">
            <button
              onClick={() => onNavigate('dashboard')}
              className="flex items-center gap-2"
            >
              <div className="w-8 h-8 bg-blue-600 rounded flex items-center justify-center">
                <svg width="18" height="18" viewBox="0 0 18 18" fill="none"
xmlns="http://www.w3.org/2000/svg">
                  <rect x="0" y="0" width="7" height="7" rx="1" fill="white" />
                  <rect x="11" y="0" width="7" height="7" rx="1" fill="white" />
                  <rect x="0" y="11" width="7" height="7" rx="1" fill="white" />
                </svg>
              </div>
              <span className="text-gray-900">DataViz</span>
            </button>

            <div className="hidden md:flex items-center gap-1">
              <button
                onClick={() => onNavigate('dashboard')}
                className={`px-3 py-2 rounded-lg transition-colors ${
                  currentPage === 'dashboard'
                    ? 'bg-gray-100 text-gray-900'
                    : 'text-gray-600 hover:bg-gray-50 hover:text-gray-900'
                }`}
```

```
            >
              Головна
            </button>
            <button
              onClick={() => onNavigate('datasets-list')}
              className={`px-3 py-2 rounded-lg transition-colors ${
                currentPage === 'datasets-list'
                  ? 'bg-gray-100 text-gray-900'
                  : 'text-gray-600 hover:bg-gray-50 hover:text-gray-900'
              }`}
            >
              Датасети
            </button>
          </div>
        </div>

        <div className="relative">
          <button
            onClick={() => setIsDropdownOpen(!isDropdownOpen)}
            className="flex items-center gap-3 px-3 py-2 rounded-lg hover:bg-gray-50 transition-colors"
          >
            <div className="w-8 h-8 bg-blue-600 rounded-full flex items-center justify-center text-white">
              {getInitials(user.name)}
            </div>
            <div className="hidden sm:block text-left">
              <div className="text-gray-900">{user.name}</div>
              <div className="text-gray-500">{user.email}</div>
            </div>
            <ChevronDown className="w-4 h-4 text-gray-400" />
          </button>

          {isDropdownOpen && (
            <>
              <div
                className="fixed inset-0 z-10"
                onClick={() => setIsDropdownOpen(false)}
              />
              <div className="absolute right-0 mt-2 w-56 bg-white rounded-lg shadow-lg border border-gray-200 py-1 z-20">
                <div className="px-4 py-3 border-b border-gray-200">
                  <div className="text-gray-900">{user.name}</div>
                  <div className="text-gray-500">{user.email}</div>
                </div>
                <button
                  onClick={() => {
                    setIsDropdownOpen(false);
                    onNavigate('profile');
                  }}
                  className="w-full px-4 py-2 text-left text-gray-700 hover:bg-gray-50 flex items-center gap-2"
```

```
                    >
                      <User className="w-4 h-4" />
                      Профіль
                    </button>
                    <button
                      onClick={() => {
                        setIsDropdownOpen(false);
                        onNavigate('profile');
                      }}
                      className="w-full px-4 py-2 text-left text-gray-700 hover:bg-gray-50 flex
        items-center gap-2"
                    >
                      <Settings className="w-4 h-4" />
                      Налаштування
                    </button>
                    <div className="border-t border-gray-200 my-1" />
                    <button
                      onClick={() => {
                        setIsDropdownOpen(false);
                          onLogout();
                          fetch('http://localhost:3001/api/logout', {
                          method: 'POST',
                          });
                      }}
                      className="w-full px-4 py-2 text-left text-red-600 hover:bg-gray-50 flex
        items-center gap-2"
                    >
                      <LogOut className="w-4 h-4" />
                      Вийти
                    </button>
                  </div>
                </>
              )}
            </div>
          </div>
        </div>
      </nav>
    );
}
```

СТОРІНКА ДАТАСЕТІВ

```
import { Plus, Upload, BarChart3 } from 'lucide-react';
import { Navbar } from '../common/Navbar';
import { DatasetCard } from './DatasetCard';
import type { User, Dataset, Page } from '../../App';

interface DashboardProps {
  user: User;
  datasets: Dataset[];
  onLogout: () => void;
  onNavigate: (page: Page, datasetId?: string) => void;
}
```

```
export function Dashboard({ user, datasets, onLogout, onNavigate }: DashboardProps) {
 return (
  <div className="min-h-screen bg-gray-50">
   <Navbar user={user} onLogout={onLogout} onNavigate={onNavigate} currentPage="dashboard" />

   <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">
    <div className="mb-8">
     <h1 className="text-gray-900 mb-2">Вітаємо, {user.name.split(' ')[0]}!</h1>
     <p className="text-gray-600">
      Керуйте своїми датасетами та аналізом даних
     </p>
    </div>

    <div className="grid grid-cols-1 md:grid-cols-3 gap-6 mb-8">
     <button
      onClick={() => onNavigate('upload-dataset')}
      className="bg-white rounded-lg border-2 border-dashed border-gray-300 p-6 hover:border-blue-500 hover:bg-blue-50 transition-all group"
     >
      <div className="flex flex-col items-center gap-3 text-center">
       <div className="w-12 h-12 bg-blue-100 rounded-lg flex items-center justify-center group-hover:bg-blue-600 transition-colors">
        <Upload className="w-6 h-6 text-blue-600 group-hover:text-white" />
       </div>
       <div>
        <div className="text-gray-900 mb-1">Завантажити датасет</div>
        <div className="text-gray-500">CSV, JSON</div>
       </div>
      </div>
     </button>

     <button
      onClick={() => onNavigate('datasets-list')}
      className="bg-white rounded-lg border border-gray-200 p-6 hover:shadow-md transition-shadow"
     >
      <div className="flex flex-col items-center gap-3 text-center">
       <div className="w-12 h-12 bg-green-100 rounded-lg flex items-center justify-center">
        <BarChart3 className="w-6 h-6 text-green-600" />
       </div>
       <div>
        <div className="text-gray-900 mb-1">Переглянути всі датасети</div>
        <div className="text-gray-500">{datasets.length} датасетів</div>
       </div>
      </div>
     </button>

     <div className="bg-gradient-to-br from-blue-600 to-blue-700 rounded-lg p-6 text-white">
      <div className="flex flex-col gap-3">
```

```
            <div className="w-12 h-12 bg-white/20 rounded-lg flex items-center justify-center">
              <Plus className="w-6 h-6 text-white" />
            </div>
            <div>
              <div className="mb-1">Новий аналіз</div>
              <div className="text-blue-100">Почніть з вибору датасету</div>
            </div>
          </div>
        </div>
      </div>

      <div className="mb-6 flex items-center justify-between">
        <h2 className="text-gray-900">Останні датасети</h2>
        <button
          onClick={() => onNavigate('datasets-list')}
          className="text-blue-600 hover:text-blue-700"
        >
          Переглянути все
        </button>
      </div>

      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
        {datasets.map((dataset) => (
          <DatasetCard
            key={dataset.id}
            dataset={dataset}
            onViewDetails={() => onNavigate('dataset-details', dataset.id)}
            onVisualize={() => onNavigate('visualization-builder', dataset.id)}
          />
        ))}
      </div>

      {datasets.length === 0 && (
        <div className="text-center py-12">
          <div className="w-16 h-16 bg-gray-100 rounded-full mx-auto mb-4 flex items-center justify-center">
            <BarChart3 className="w-8 h-8 text-gray-400" />
          </div>
          <h3 className="text-gray-900 mb-2">Немає датасетів</h3>
          <p className="text-gray-600 mb-6">
            Завантажте свій перший датасет для початку аналізу
          </p>
          <button
            onClick={() => onNavigate('upload-dataset')}
            className="bg-blue-600 text-white px-6 py-2.5 rounded-lg hover:bg-blue-700 transition-colors inline-flex items-center gap-2"
          >
            <Upload className="w-4 h-4" />
            Завантажити датасет
          </button>
        </div>
```

```
            )}
          </div>
        </div>
      );
}
```

СТОРІНКА КАРТОЧК ДАТАСЕТУ

```
import { LineChart, Line, ResponsiveContainer } from 'recharts';
import { Database, FileText, Clock, BarChart3, AlertCircle } from 'lucide-react';
import type { Dataset } from '../../App';

interface DatasetCardProps {
  dataset: Dataset;
  onViewDetails: () => void;
  onVisualize: () => void;
}

export function DatasetCard({ dataset, onViewDetails, onVisualize }: DatasetCardProps) {
  // Mock chart data
  const [chartData, setChartData] = useState<{ value: number }[]>([]);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch(`http://localhost:3001/api/datasets/${dataset.id}/data`);
        const data = await response.json();
        setChartData(data);
      } catch (error) {
        console.error('Error fetching chart data:', error);
      }
    };

    fetchData();
  }, [dataset.id]);

  const getStatusColor = (status: string) => {
    switch (status) {
      case 'ready':
        return 'bg-green-100 text-green-700';
      case 'processing':
        return 'bg-yellow-100 text-yellow-700';
      case 'error':
        return 'bg-red-100 text-red-700';
      default:
        return 'bg-gray-100 text-gray-700';
    }
  };

  const getStatusText = (status: string) => {
    switch (status) {
      case 'ready':
        return 'Готово';
      case 'processing':
```

```
      return 'Обробка';
    case 'error':
      return 'Помилка';
    default:
      return status;
  }
 };

 return (
  <div className="bg-white rounded-lg border border-gray-200 hover:shadow-md transition-shadow overflow-hidden group">
    <div className="p-6">
     <div className="flex items-start justify-between mb-4">
      <div className="flex-1">
       <h3 className="text-gray-900 mb-1 group-hover:text-blue-600 transition-colors">
        {dataset.name}
       </h3>
       <p className="text-gray-500">{dataset.description}</p>
      </div>
      <span className={`px-2 py-1 rounded text-xs ${getStatusColor(dataset.status)}`}>
       {getStatusText(dataset.status)}
      </span>
     </div>

     <div className="grid grid-cols-3 gap-4 mb-4">
      <div className="flex items-center gap-2">
       <Database className="w-4 h-4 text-gray-400" />
       <div>
        <div className="text-gray-500">Рядки</div>
        <div className="text-gray-900">{dataset.rows.toLocaleString()}</div>
       </div>
      </div>
      <div className="flex items-center gap-2">
       <FileText className="w-4 h-4 text-gray-400" />
       <div>
        <div className="text-gray-500">Колонки</div>
        <div className="text-gray-900">{dataset.columns}</div>
       </div>
      </div>
      <div className="flex items-center gap-2">
       <Clock className="w-4 h-4 text-gray-400" />
       <div>
        <div className="text-gray-500">Формат</div>
        <div className="text-gray-900">{dataset.format}</div>
       </div>
      </div>
     </div>

     <div className="h-16 mb-4 bg-gray-50 rounded-lg overflow-hidden">
      {dataset.status === 'ready' ? (
       <ResponsiveContainer width="100%" height="100%">
        <LineChart data={chartData}>
```

```
                    <Line
                      type="monotone"
                      dataKey="value"
                      stroke="#3b82f6"
                      strokeWidth={2}
                      dot={false}
                    />
                  </LineChart>
                </ResponsiveContainer>
              ) : dataset.status === 'processing' ? (
                <div className="h-full flex items-center justify-center">
                  <div className="flex items-center gap-2 text-yellow-600">
                    <Clock className="w-4 h-4 animate-spin" />
                    <span>Обробка...</span>
                  </div>
                </div>
              ) : (
                <div className="h-full flex items-center justify-center">
                  <div className="flex items-center gap-2 text-red-600">
                    <AlertCircle className="w-4 h-4" />
                    <span>Помилка обробки</span>
                  </div>
                </div>
              )}
            </div>

            <div className="flex gap-2">
              <button
                onClick={onViewDetails}
                className="flex-1 px-4 py-2 border border-gray-300 rounded-lg text-gray-700 hover:bg-gray-50 transition-colors"
              >
                Деталі
              </button>
              {dataset.status === 'ready' && (
                <button
                  onClick={onVisualize}
                  className="flex-1 px-4 py-2 bg-blue-600 text-white rounded-lg hover:bg-blue-700 transition-colors flex items-center justify-center gap-2"
                >
                  <BarChart3 className="w-4 h-4" />
                  Аналіз
                </button>
              )}
            </div>
          </div>
        </div>
      );
}
```

СТОРІНКА ДЕТАЛЕЙ ДАТАСЕТУ

```
import { useState } from 'react';
import { ArrowLeft, Download, BarChart3, Database, Filter } from 'lucide-react';
```

```
import { Navbar } from '../common/Navbar';
import type { User, Dataset, Page } from '../../App';

interface DatasetDetailsProps {
  user: User;
  dataset: Dataset;
  onLogout: () => void;
  onNavigate: (page: Page, datasetId?: string) => void;
}

export function DatasetDetails({ user, dataset, onLogout, onNavigate }: DatasetDetailsProps) {
  const [activeTab, setActiveTab] = useState<'preview' | 'stats' | 'preprocessing'>('preview');

  const [mockData, setMockData] = useState<any[]>([]);

  useState(() => {
    // Fetch data from local REST service
    const fetchDatasetPreview = async () => {
      try {
        const response = await fetch(`http://localhost:8000/api/datasets/${dataset.id}/preview`);
        if (response.ok) {
          const data = await response.json();
          setMockData(data);
        }
      } catch (error) {
        console.error('Failed to fetch dataset preview:', error);
      }
    };

    fetchDatasetPreview();
  }, [dataset.id]);

  const columns = dataset.columns_list || ['ID', 'Регіон', 'Продукт', 'Кількість', 'Дохід', 'Витрати'];

  const [stats, setStats] = useState<Array<{ name: string; value: string }>>([]);

  useState(() => {
    const fetchDatasetStats = async () => {
      try {
        const response = await fetch(`http://localhost:8000/api/datasets/${dataset.id}/stats`);
        if (response.ok) {
          const data = await response.json();
          setStats([
            { name: 'Загальна кількість рядків', value: data.rows?.toLocaleString() || dataset.rows.toLocaleString() },
            { name: 'Кількість колонок', value: data.columns || dataset.columns },
            { name: 'Розмір файлу', value: data.size || dataset.size },
            { name: 'Формат', value: data.format || dataset.format },
            { name: 'Відсутні значення', value: data.missing_values || '0.5%' },
```

```
          { name: 'Дублікати', value: data.duplicates || '12 рядків' },
        ]);
       }
      } catch (error) {
       console.error('Failed to fetch dataset stats:', error);
       setStats([
        { name: 'Загальна кількість рядків', value: dataset.rows.toLocaleString() },
        { name: 'Кількість колонок', value: dataset.columns },
        { name: 'Розмір файлу', value: dataset.size },
        { name: 'Формат', value: dataset.format },
        { name: 'Відсутні значення', value: '0.5%' },
        { name: 'Дублікати', value: '12 рядків' },
       ]);
      }
     };

     fetchDatasetStats();
    }, [dataset.id]);

    return (
     <div className="min-h-screen bg-gray-50">
      <Navbar user={user} onLogout={onLogout} onNavigate={onNavigate} />

      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">
       <button
        onClick={() => onNavigate('dashboard')}
        className="flex items-center gap-2 text-gray-600 hover:text-gray-900 mb-6"
       >
        <ArrowLeft className="w-4 h-4" />
        Назад до датасетів
       </button>

       <div className="bg-white rounded-lg border border-gray-200 mb-6">
        <div className="p-6 border-b border-gray-200">
         <div className="flex items-start justify-between mb-4">
          <div className="flex-1">
           <h1 className="text-gray-900 mb-2">{dataset.name}</h1>
           <p className="text-gray-600">{dataset.description}</p>
          </div>
          <div className="flex gap-2">
           <button className="px-4 py-2 border border-gray-300 rounded-lg text-gray-700
hover:bg-gray-50 transition-colors inline-flex items-center gap-2">
            <Download className="w-4 h-4" />
            Експорт
           </button>
           <button
            onClick={() => onNavigate('visualization-builder', dataset.id)}
            className="px-4 py-2 bg-blue-600 text-white rounded-lg hover:bg-blue-700
transition-colors inline-flex items-center gap-2"
           >
            <BarChart3 className="w-4 h-4" />
            Запустити аналіз
```

```
      </button>
    </div>
  </div>

  <div className="grid grid-cols-2 md:grid-cols-4 gap-4">
    <div className="bg-gray-50 rounded-lg p-4">
      <div className="text-gray-600 mb-1">Рядки</div>
      <div className="text-gray-900">{dataset.rows.toLocaleString()}</div>
    </div>
    <div className="bg-gray-50 rounded-lg p-4">
      <div className="text-gray-600 mb-1">Колонки</div>
      <div className="text-gray-900">{dataset.columns}</div>
    </div>
    <div className="bg-gray-50 rounded-lg p-4">
      <div className="text-gray-600 mb-1">Розмір</div>
      <div className="text-gray-900">{dataset.size}</div>
    </div>
    <div className="bg-gray-50 rounded-lg p-4">
      <div className="text-gray-600 mb-1">Формат</div>
      <div className="text-gray-900">{dataset.format}</div>
    </div>
  </div>
</div>

<div className="border-b border-gray-200">
  <div className="flex px-6">
    <button
      onClick={() => setActiveTab('preview')}
      className={`px-4 py-3 border-b-2 transition-colors ${
        activeTab === 'preview'
          ? 'border-blue-600 text-blue-600'
          : 'border-transparent text-gray-600 hover:text-gray-900'
      }`}
    >
      Попередній перегляд
    </button>
    <button
      onClick={() => setActiveTab('stats')}
      className={`px-4 py-3 border-b-2 transition-colors ${
        activeTab === 'stats'
          ? 'border-blue-600 text-blue-600'
          : 'border-transparent text-gray-600 hover:text-gray-900'
      }`}
    >
      Статистика
    </button>
    <button
      onClick={() => setActiveTab('preprocessing')}
      className={`px-4 py-3 border-b-2 transition-colors ${
        activeTab === 'preprocessing'
          ? 'border-blue-600 text-blue-600'
          : 'border-transparent text-gray-600 hover:text-gray-900'
```

```
            }`}
           >
            Попередня обробка
           </button>
          </div>
         </div>

         <div className="p-6">
          {activeTab === 'preview' && (
           <div>
            <div className="flex items-center justify-between mb-4">
             <h3 className="text-gray-900">Перегляд даних</h3>
             <button className="px-3 py-1.5 border border-gray-300 rounded-lg text-gray-
700 hover:bg-gray-50 transition-colors inline-flex items-center gap-2">
              <Filter className="w-4 h-4" />
              Фільтри
             </button>
            </div>
            <div className="overflow-x-auto border border-gray-200 rounded-lg">
             <table className="w-full">
              <thead className="bg-gray-50">
               <tr>
                {columns.map((col, idx) => (
                 <th key={idx} className="px-4 py-3 text-left text-gray-700 border-b
border-gray-200">
                  {col}
                 </th>
                ))}
               </tr>
              </thead>
              <tbody className="divide-y divide-gray-200">
               {mockData.map((row, idx) => (
                <tr key={idx} className="hover:bg-gray-50">
                 <td className="px-4 py-3 text-gray-900">{row.id}</td>
                 <td className="px-4 py-3 text-gray-900">{row.region}</td>
                 <td className="px-4 py-3 text-gray-900">{row.product}</td>
                 <td className="px-4 py-3 text-gray-900">{row.quantity}</td>
                 <td                 className="px-4                 py-3                 text-gray-
900">{row.revenue.toLocaleString()}</td>
                 <td                 className="px-4                 py-3                 text-gray-
900">{row.expenses.toLocaleString()}</td>
                </tr>
               ))}
              </tbody>
             </table>
            </div>
            <div className="mt-4 text-center text-gray-600">
             Показано 5 з {dataset.rows.toLocaleString()} рядків
            </div>
           </div>
          )}
```

```
        {activeTab === 'stats' && (
          <div>
            <h3 className="text-gray-900 mb-4">Базова статистика колонок</h3>
            <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
              {stats.map((stat, idx) => (
                <div key={idx} className="border border-gray-200 rounded-lg p-4">
                  <div className="text-gray-600 mb-2">{stat.name}</div>
                  <div className="text-gray-900">{stat.value}</div>
                </div>
              ))}
            </div>
          </div>
        )}

        {activeTab === 'preprocessing' && (
          <div>
            <h3 className="text-gray-900 mb-4">Інструменти попередньої обробки</h3>
            <div className="space-y-4">
              <div className="border border-gray-200 rounded-lg p-4">
                <h4 className="text-gray-900 mb-2">Заповнити відсутні значення</h4>
                <p className="text-gray-600 mb-4">
                  Виберіть метод заповнення пропущених значень
                </p>
                <select className="w-full px-4 py-2 border border-gray-300 rounded-lg
focus:outline-none focus:ring-2 focus:ring-blue-500">
                  <option>Середнє значення</option>
                  <option>Медіана</option>
                  <option>Найбільш частне значення</option>
                  <option>Нуль</option>
                </select>
              </div>

              <div className="border border-gray-200 rounded-lg p-4">
                <h4 className="text-gray-900 mb-2">Нормалізація даних</h4>
                <p className="text-gray-600 mb-4">
                  Застосувати нормалізацію до числових колонок
                </p>
                <select className="w-full px-4 py-2 border border-gray-300 rounded-lg
focus:outline-none focus:ring-2 focus:ring-blue-500">
                  <option>Min-Max нормалізація</option>
                  <option>Z-score стандартизація</option>
                  <option>Робастна нормалізація</option>
                </select>
              </div>

              <div className="flex gap-3">
                <button className="flex-1 px-6 py-2.5 border border-gray-300 rounded-lg
text-gray-700 hover:bg-gray-50 transition-colors">
                  Скасувати
                </button>
                <button className="flex-1 px-6 py-2.5 bg-blue-600 text-white rounded-lg
hover:bg-blue-700 transition-colors">
```

```
                    Застосувати
                  </button>
                </div>
              </div>
            </div>
          )}
        </div>
      </div>
    </div>
  </div>
 );
}
```

ФОРМА СПИСКУ ДАТАСЕТІВ

```
import { useState } from 'react';
import { Upload, Search, Download, Trash2, Filter, ChevronDown, ChevronUp } from
'lucide-react';
import { Navbar } from '../common/Navbar';
import type { User, Dataset, Page } from '../../App';

interface DatasetsListProps {
  user: User;
  datasets: Dataset[];
  onLogout: () => void;
  onNavigate: (page: Page, datasetId?: string) => void;
  onDelete: (id: string) => void;
}

export function DatasetsList({ user, datasets, onLogout, onNavigate, onDelete }:
DatasetsListProps) {
  const [searchQuery, setSearchQuery] = useState('');
  const [sortBy, setSortBy] = useState<'name' | 'date' | 'size'>('date');
  const [expandedRow, setExpandedRow] = useState<string | null>(null);

  const filteredDatasets = datasets.filter(dataset =>
    dataset.name.toLowerCase().includes(searchQuery.toLowerCase()) ||
    dataset.description?.toLowerCase().includes(searchQuery.toLowerCase())
  );

  const sortedDatasets = [...filteredDatasets].sort((a, b) => {
    if (sortBy === 'name') return a.name.localeCompare(b.name);
    if (sortBy === 'date') return new Date(b.lastUpdated).getTime() - new
Date(a.lastUpdated).getTime();
    if (sortBy === 'size') return parseFloat(b.size) - parseFloat(a.size);
    return 0;
  });

  const handleDelete = (id: string, name: string) => {
    if (window.confirm(`Ви впевнені, що хочете видалити датасет "${name}"?`)) {
      try {
        const response = await fetch(`/api/datasets/${id}`, {
          method: 'DELETE',
        });
```

```
      if (response.ok) {
        onDelete(id);
      } else {
        alert('Не вдалося видалити датасет');
      }
    } catch (error) {
      console.error('Error deleting dataset:', error);
      alert('Помилка при видаленні датасету');
    }
  }
};

const formatDate = (dateString: string) => {
  const date = new Date(dateString);
  return date.toLocaleDateString('uk-UA', {
    year: 'numeric',
    month: 'long',
    day: 'numeric'
  });
};

const getStatusColor = (status: string) => {
  switch (status) {
    case 'ready':
      return 'bg-green-100 text-green-700';
    case 'processing':
      return 'bg-yellow-100 text-yellow-700';
    case 'error':
      return 'bg-red-100 text-red-700';
    default:
      return 'bg-gray-100 text-gray-700';
  }
};

const getStatusText = (status: string) => {
  switch (status) {
    case 'ready':
      return 'Готово';
    case 'processing':
      return 'Обробка';
    case 'error':
      return 'Помилка';
    default:
      return status;
  }
};

return (
  <div className="min-h-screen bg-gray-50">
    <Navbar user={user} onLogout={onLogout} onNavigate={onNavigate}
currentPage="datasets-list" />
```

```
    <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">
      <div className="flex items-center justify-between mb-8">
        <div>
          <h1 className="text-gray-900 mb-2">Датасети</h1>
          <p className="text-gray-600">
            Управління вашими датасетами
          </p>
        </div>
        <button
          onClick={() => onNavigate('upload-dataset')}
          className="bg-blue-600 text-white px-6 py-2.5 rounded-lg hover:bg-blue-700 transition-colors inline-flex items-center gap-2"
        >
          <Upload className="w-4 h-4" />
          Завантажити датасет
        </button>
      </div>

      <div className="bg-white rounded-lg border border-gray-200">
        <div className="p-4 border-b border-gray-200">
          <div className="flex flex-col md:flex-row gap-4">
            <div className="flex-1 relative">
              <Search className="absolute left-3 top-1/2 -translate-y-1/2 w-5 h-5 text-gray-400" />
              <input
                type="text"
                value={searchQuery}
                onChange={(e) => setSearchQuery(e.target.value)}
                placeholder="Пошук датасетів..."
                className="w-full pl-10 pr-4 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent"
              />
            </div>
            <div className="flex gap-2">
              <select
                value={sortBy}
                onChange={(e) => setSortBy(e.target.value as 'name' | 'date' | 'size')}
                className="px-4 py-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent"
              >
                <option value="date">Сортувати за датою</option>
                <option value="name">Сортувати за назвою</option>
                <option value="size">Сортувати за розміром</option>
              </select>
              <button className="px-4 py-2 border border-gray-300 rounded-lg hover:bg-gray-50 transition-colors">
                <Filter className="w-5 h-5 text-gray-600" />
              </button>
            </div>
          </div>
        </div>
```

```
        <div className="overflow-x-auto">
          <table className="w-full">
            <thead className="bg-gray-50 border-b border-gray-200">
              <tr>
                <th className="px-6 py-3 text-left text-gray-700">Назва</th>
                <th className="px-6 py-3 text-left text-gray-700">Розмір</th>
                <th className="px-6 py-3 text-left text-gray-700">Формат</th>
                <th className="px-6 py-3 text-left text-gray-700">Статус</th>
                <th className="px-6 py-3 text-left text-gray-700">Оновлено</th>
                <th className="px-6 py-3 text-right text-gray-700">Дії</th>
              </tr>
            </thead>
            <tbody className="divide-y divide-gray-200">
              {sortedDatasets.map((dataset) => (
                <>
                  <tr key={dataset.id} className="hover:bg-gray-50 transition-colors">
                    <td className="px-6 py-4">
                      <div>
                        <div className="text-gray-900">{dataset.name}</div>
                        <div className="text-gray-500">
                          {dataset.rows.toLocaleString()} рядків, {dataset.columns} колонок
                        </div>
                      </div>
                    </td>
                    <td className="px-6 py-4 text-gray-900">{dataset.size}</td>
                    <td className="px-6 py-4">
                      <span className="px-2 py-1 bg-gray-100 text-gray-700 rounded">
                        {dataset.format}
                      </span>
                    </td>
                    <td className="px-6 py-4">
                      <span className={`px-2 py-1 rounded text-xs
${getStatusColor(dataset.status)}`}>
                        {getStatusText(dataset.status)}
                      </span>
                    </td>
                    <td className="px-6 py-4 text-gray-900">
                      {formatDate(dataset.lastUpdated)}
                    </td>
                    <td className="px-6 py-4">
                      <div className="flex items-center justify-end gap-2">
                        <button
                          onClick={() => setExpandedRow(expandedRow === dataset.id ? null :
dataset.id)}
                          className="p-2 text-gray-600 hover:bg-gray-100 rounded-lg transition-
colors"
                          title="Переглянути перші рядки"
                        >
                          {expandedRow === dataset.id ? (
                            <ChevronUp className="w-4 h-4" />
                          ) : (
                            <ChevronDown className="w-4 h-4" />
```

```
                    )}
                  </button>
                  <button
                    onClick={() => onNavigate('dataset-details', dataset.id)}
                    className="px-3 py-1.5 text-blue-600 hover:bg-blue-50 rounded-lg transition-colors"
                  >
                    Деталі
                  </button>
                  <button className="p-2 text-gray-600 hover:bg-gray-100 rounded-lg transition-colors">
                    <Download className="w-4 h-4" />
                  </button>
                  <button
                    onClick={() => handleDelete(dataset.id, dataset.name)}
                    className="p-2 text-red-600 hover:bg-red-50 rounded-lg transition-colors"
                  >
                    <Trash2 className="w-4 h-4" />
                  </button>
                </div>
              </td>
            </tr>
            {expandedRow === dataset.id && (
              <tr>
                <td colSpan={6} className="px-6 py-4 bg-gray-50">
                  <div className="space-y-2">
                    <div  className="text-gray-700">Попередній перегляд перших рядків:</div>
                    <div className="bg-white rounded border border-gray-200 p-4 overflow-x-auto">
                      <div className="text-gray-600">
                        Колонки: {dataset.columns_list?.join(', ') || 'Дані завантажуються...'}
                      </div>
                    </div>
                  </div>
                </td>
              </tr>
            )}
          </>
        ))}
      </tbody>
    </table>
  </div>

  {sortedDatasets.length === 0 && (
    <div className="text-center py-12">
      <p className="text-gray-600">
        {searchQuery ? 'Датасети не знайдено' : 'Немає датасетів'}
      </p>
    </div>
  )}
```

```
            </div>
          </div>
        </div>
      );
}
```

ФОРМА ЗАВАНТАЖЕННЯ ДАТАСЕТУ

```
import { useState } from 'react';
import { Upload, X, FileText, CheckCircle } from 'lucide-react';
import { Navbar } from '../common/Navbar';
import type { User, Dataset, Page } from '../../App';

interface UploadDatasetProps {
  user: User;
  onLogout: () => void;
  onNavigate: (page: Page) => void;
  onUpload: (dataset: Dataset) => void;
}

export function UploadDataset({ user, onLogout, onNavigate, onUpload }:
UploadDatasetProps) {
  const [dragActive, setDragActive] = useState(false);
  const [file, setFile] = useState<File | null>(null);
  const [formData, setFormData] = useState({
    name: '',
    description: ''
  });
  const [uploading, setUploading] = useState(false);
  const [uploadProgress, setUploadProgress] = useState(0);

  const handleDrag = (e: React.DragEvent) => {
    e.preventDefault();
    e.stopPropagation();
    if (e.type === "dragenter" || e.type === "dragover") {
      setDragActive(true);
    } else if (e.type === "dragleave") {
      setDragActive(false);
    }
  };

  const handleDrop = (e: React.DragEvent) => {
    e.preventDefault();
    e.stopPropagation();
    setDragActive(false);

    if (e.dataTransfer.files && e.dataTransfer.files[0]) {
      handleFile(e.dataTransfer.files[0]);
    }
  };

  const handleFileInput = (e: React.ChangeEvent<HTMLInputElement>) => {
    if (e.target.files && e.target.files[0]) {
      handleFile(e.target.files[0]);
```

```
  }
 };

 const handleFile = (file: File) => {
  const validFormats = ['text/csv', 'application/json'];
  if (validFormats.includes(file.type) || file.name.endsWith('.csv') ||
file.name.endsWith('.json')) {
   setFile(file);
   if (!formData.name) {
   setFormData({ ...formData, name: file.name.replace(/\.[^/.]+$/, '') });
   }
  }
  };

  const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();
  if (!file || !formData.name) return;

  setUploading(true);
  setUploadProgress(0);

  try {
   const data = new FormData();
   data.append('file', file);
   data.append('name', formData.name);
   data.append('description', formData.description);

   const response = await fetch('http://localhost:3001/api/datasets/upload', {
   method: 'POST',
   body: data
   });

   if (!response.ok) throw new Error('Upload failed');

   const newDataset: Dataset = await response.json();
   onUpload(newDataset);
   setUploading(false);
  } catch (error) {
   console.error('Upload error:', error);
   setUploading(false);
  }
  };

  return (
  <div className="min-h-screen bg-gray-50">
   <Navbar user={user} onLogout={onLogout} onNavigate={onNavigate} />

   <div className="max-w-4xl mx-auto px-4 sm:px-6 lg:px-8 py-8">
   <div className="mb-8">
    <h1 className="text-gray-900 mb-2">Завантажити датасет</h1>
    <p className="text-gray-600">
    Завантажте ваш датасет у форматі CSV або JSON
```

```
        </p>
      </div>

      <div className="bg-white rounded-lg border border-gray-200 p-8">
        <form onSubmit={handleSubmit} className="space-y-6">
          <div
            onDragEnter={handleDrag}
            onDragLeave={handleDrag}
            onDragOver={handleDrag}
            onDrop={handleDrop}
            className={`relative border-2 border-dashed rounded-lg p-12 text-center transition-colors ${
              dragActive
                ? 'border-blue-500 bg-blue-50'
                : file
                ? 'border-green-500 bg-green-50'
                : 'border-gray-300 hover:border-gray-400'
            }`}
          >
            <input
              type="file"
              id="file-upload"
              accept=".csv,.json"
              onChange={handleFileInput}
              className="hidden"
            />

            {!file ? (
              <>
                <Upload className="w-12 h-12 text-gray-400 mx-auto mb-4" />
                <h3 className="text-gray-900 mb-2">
                  Перетягніть файл сюди або клацніть для вибору
                </h3>
                <p className="text-gray-600 mb-4">
                  Підтримувані формати: CSV, JSON
                </p>
                <label
                  htmlFor="file-upload"
                  className="inline-block bg-blue-600 text-white px-6 py-2.5 rounded-lg hover:bg-blue-700 transition-colors cursor-pointer"
                >
                  Вибрати файл
                </label>
              </>
            ) : (
              <div className="flex items-center justify-center gap-4">
                <FileText className="w-8 h-8 text-green-600" />
                <div className="text-left">
                  <div className="text-gray-900">{file.name}</div>
                  <div className="text-gray-500">
                    {(file.size / 1024 / 1024).toFixed(2)} MB
                  </div>
```

```
                </div>
                <button
                  type="button"
                  onClick={() => setFile(null)}
                  className="p-2 hover:bg-white rounded-lg transition-colors"
                >
                  <X className="w-5 h-5 text-gray-400" />
                </button>
              </div>
            )}
          </div>

          {uploading && (
            <div className="space-y-2">
              <div className="flex items-center justify-between">
                <span className="text-gray-700">Завантаження...</span>
                <span className="text-gray-900">{uploadProgress}%</span>
              </div>
              <div className="w-full bg-gray-200 rounded-full h-2 overflow-hidden">
                <div
                  className="bg-blue-600 h-full transition-all duration-300"
                  style={{ width: `${uploadProgress}%` }}
                />
              </div>
            </div>
          )}

          <div>
            <label htmlFor="name" className="block text-gray-700 mb-2">
              Назва датасету *
            </label>
            <input
              id="name"
              type="text"
              value={formData.name}
              onChange={(e) => setFormData({ ...formData, name: e.target.value })}
              className="w-full px-4 py-2.5 border border-gray-300 rounded-lg focus:outline-
none focus:ring-2 focus:ring-blue-500 focus:border-transparent"
              placeholder="Введіть назву датасету"
              required
            />
          </div>

          <div>
            <label htmlFor="description" className="block text-gray-700 mb-2">
              Опис
            </label>
            <textarea
              id="description"
              value={formData.description}
              onChange={(e) => setFormData({ ...formData, description: e.target.value })}
```

```
                    className="w-full px-4 py-2.5 border border-gray-300 rounded-lg focus:outline-
        none focus:ring-2 focus:ring-blue-500 focus:border-transparent resize-none"
                    placeholder="Додайте опис датасету"
                    rows={3}
                  />
                </div>

                <div className="flex gap-3 pt-4">
                  <button
                    type="button"
                    onClick={() => onNavigate('dashboard')}
                    className="flex-1 px-6 py-2.5 border border-gray-300 rounded-lg text-gray-700
        hover:bg-gray-50 transition-colors"
                  >
                    Скасувати
                  </button>
                  <button
                    type="submit"
                    disabled={!file || !formData.name || uploading}
                    className="flex-1 px-6 py-2.5 bg-blue-600 text-white rounded-lg hover:bg-blue-
        700 transition-colors disabled:bg-gray-300 disabled:cursor-not-allowed flex items-center
        justify-center gap-2"
                  >
                    {uploading ? (
                      <>
                        <div className="w-4 h-4 border-2 border-white border-t-transparent rounded-
        full animate-spin" />
                        Завантаження...
                      </>
                    ) : (
                      <>
                        <Upload className="w-4 h-4" />
                        Завантажити датасет
                      </>
                    )}
                  </button>
                </div>
              </form>
            </div>
          </div>
        </div>
      );
}
```

ФОРМА РЕДАГУВАННЯ ПРОФІЛЮ

```
import { useState } from 'react';
import { User as UserIcon, Mail, Building, MapPin, Globe, Save } from 'lucide-react';
import { Navbar } from '../common/Navbar';
import type { User, Page } from '../../App';

interface UserProfileProps {
  user: User;
```

```
  onLogout: () => void;
  onNavigate: (page: Page) => void;
  onUpdateUser: (user: User) => void;
}

export function UserProfile({ user, onLogout, onNavigate, onUpdateUser }:
UserProfileProps) {
  const [formData, setFormData] = useState<User>(user);
  const [isSaved, setIsSaved] = useState(false);

  const handleChange = (field: keyof User, value: string) => {
    setFormData({ ...formData, [field]: value });
  };

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    onUpdateUser(formData);
    setIsSaved(true);
    fetch('http://localhost:3001/api/user', {
      method: 'PUT',
      headers: {
      'Content-Type': 'application/json',
      },
      body: JSON.stringify(formData),
    })
    .then(response => {
      if (!response.ok) {
      throw new Error('Network response was not ok');
      }
      return response.json();
    })
    .then(data => {
      console.log('User updated:', data);
      setIsSaved(true);
    })
    .catch(error => {
      console.error('There was a problem with the fetch operation:', error);
    });
  };

  return (
    <div className="min-h-screen bg-gray-50">
      <Navbar user={user} onLogout={onLogout} onNavigate={onNavigate}
currentPage="profile" />

      <div className="max-w-4xl mx-auto px-4 sm:px-6 lg:px-8 py-8">
        <div className="mb-8">
          <h1 className="text-gray-900 mb-2">Профіль користувача</h1>
          <p className="text-gray-600">
            Керуйте своєю особистою інформацією
          </p>
        </div>
```

```
      <div className="grid grid-cols-1 lg:grid-cols-3 gap-6">
        {/* Profile sidebar */}
        <div className="lg:col-span-1">
          <div className="bg-white rounded-lg border border-gray-200 p-6">
            <div className="text-center mb-6">
              <div className="w-24 h-24 bg-blue-600 rounded-full mx-auto mb-4 flex items-center justify-center text-white">
                <span className="text-3xl">
                  {user.name.split(' ').map(n => n[0]).join('').toUpperCase().slice(0, 2)}
                </span>
              </div>
              <h3 className="text-gray-900 mb-1">{user.name}</h3>
              <p className="text-gray-600">{user.email}</p>
            </div>

            <div className="space-y-3 border-t border-gray-200 pt-6">
              <div className="flex items-center gap-3 text-gray-600">
                <Mail className="w-4 h-4" />
                <span>{user.email}</span>
              </div>
              {user.company && (
                <div className="flex items-center gap-3 text-gray-600">
                  <Building className="w-4 h-4" />
                  <span>{user.company}</span>
                </div>
              )}
              {user.city && (
                <div className="flex items-center gap-3 text-gray-600">
                  <MapPin className="w-4 h-4" />
                  <span>{user.city}, {user.country}</span>
                </div>
              )}
            </div>
          </div>
        </div>

        {/* Profile form */}
        <div className="lg:col-span-2">
          <div className="bg-white rounded-lg border border-gray-200 p-6">
            <h3 className="text-gray-900 mb-6">Особиста інформація</h3>

            {isSaved && (
              <div className="mb-6 p-4 bg-green-50 border border-green-200 rounded-lg flex items-center gap-2 text-green-700">
                <Save className="w-5 h-5" />
                <span>Зміни успішно збережено</span>
              </div>
            )}

            <form onSubmit={handleSubmit} className="space-y-5">
              <div className="grid grid-cols-1 md:grid-cols-2 gap-5">
```

```
            <div>
              <label htmlFor="name" className="block text-gray-700 mb-2">
                Повне ім'я *
              </label>
              <div className="relative">
                <UserIcon className="absolute left-3 top-1/2 -translate-y-1/2 w-5 h-5 text-
gray-400" />
                <input
                  id="name"
                  type="text"
                  value={formData.name}
                  onChange={(e) => handleChange('name', e.target.value)}
                  className="w-full pl-10 pr-4 py-2.5 border border-gray-300 rounded-lg
focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent"
                  required
                />
              </div>
            </div>

            <div>
              <label htmlFor="email" className="block text-gray-700 mb-2">
                Email *
              </label>
              <div className="relative">
                <Mail className="absolute left-3 top-1/2 -translate-y-1/2 w-5 h-5 text-gray-
400" />
                <input
                  id="email"
                  type="email"
                  value={formData.email}
                  onChange={(e) => handleChange('email', e.target.value)}
                  className="w-full pl-10 pr-4 py-2.5 border border-gray-300 rounded-lg
focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent"
                  required
                />
              </div>
            </div>
          </div>

          <div>
            <label htmlFor="company" className="block text-gray-700 mb-2">
              Компанія
            </label>
            <div className="relative">
              <Building className="absolute left-3 top-1/2 -translate-y-1/2 w-5 h-5 text-
gray-400" />
              <input
                id="company"
                type="text"
                value={formData.company || ''}
                onChange={(e) => handleChange('company', e.target.value)}
```

```
                  className="w-full pl-10 pr-4 py-2.5 border border-gray-300 rounded-lg
focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent"
                  placeholder="Назва компанії"
                 />
                </div>
               </div>

               <div className="grid grid-cols-1 md:grid-cols-2 gap-5">
                <div>
                 <label htmlFor="city" className="block text-gray-700 mb-2">
                  Місто
                 </label>
                 <div className="relative">
                  <MapPin className="absolute left-3 top-1/2 -translate-y-1/2 w-5 h-5 text-
gray-400" />
                  <input
                   id="city"
                   type="text"
                   value={formData.city || ''}
                   onChange={(e) => handleChange('city', e.target.value)}
                   className="w-full pl-10 pr-4 py-2.5 border border-gray-300 rounded-lg
focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent"
                   placeholder="Київ"
                  />
                 </div>
                </div>

                <div>
                 <label htmlFor="country" className="block text-gray-700 mb-2">
                  Країна
                 </label>
                 <div className="relative">
                  <Globe className="absolute left-3 top-1/2 -translate-y-1/2 w-5 h-5 text-gray-
400" />
                  <input
                   id="country"
                   type="text"
                   value={formData.country || ''}
                   onChange={(e) => handleChange('country', e.target.value)}
                   className="w-full pl-10 pr-4 py-2.5 border border-gray-300 rounded-lg
focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent"
                   placeholder="Україна"
                  />
                 </div>
                </div>
               </div>

               <div className="flex gap-3 pt-4">
                <button
                 type="button"
                 onClick={() => onNavigate('dashboard')}
```

```
                  className="flex-1 px-6 py-2.5 border border-gray-300 rounded-lg text-gray-700 hover:bg-gray-50 transition-colors"
                >
                  Скасувати
                </button>
                <button
                  type="submit"
                  className="flex-1 px-6 py-2.5 bg-blue-600 text-white rounded-lg hover:bg-blue-700 transition-colors flex items-center justify-center gap-2"
                >
                  <Save className="w-4 h-4" />
                  Зберегти зміни
                </button>
              </div>
            </form>
          </div>

          {/* Additional sections */}
          <div className="mt-6 bg-white rounded-lg border border-gray-200 p-6">
            <h3 className="text-gray-900 mb-4">Безпека акаунту</h3>
            <div className="space-y-4">
              <button className="w-full text-left px-4 py-3 border border-gray-300 rounded-lg hover:bg-gray-50 transition-colors flex items-center justify-between">
                <span className="text-gray-700">Змінити пароль</span>
                <span className="text-gray-400">{'>'}</span>
              </button>
              <button className="w-full text-left px-4 py-3 border border-gray-300 rounded-lg hover:bg-gray-50 transition-colors flex items-center justify-between">
                <span className="text-gray-700">Двофакторна автентифікація</span>
                <span className="px-2 py-1 bg-gray-100 text-gray-600 rounded">Вимкнено</span>
              </button>
            </div>
          </div>

          <div className="mt-6 bg-white rounded-lg border border-red-200 p-6">
            <h3 className="text-red-900 mb-2">Небезпечна зона</h3>
            <p className="text-gray-600 mb-4">
              Видалення акаунту є незворотною дією. Всі ваші дані будуть втрачені.
            </p>
            <button className="px-4 py-2 border border-red-300 text-red-600 rounded-lg hover:bg-red-50 transition-colors">
              Видалити акаунт
            </button>
          </div>
        </div>
      </div>
    </div>
  );
```

```
}
СТИЛІ
@custom-variant dark (&:is(.dark *));

:root {
  --font-size: 16px;
  --background: #ffffff;
  --foreground: oklch(0.145 0 0);
  --card: #ffffff;
  --card-foreground: oklch(0.145 0 0);
  --popover: oklch(1 0 0);
  --popover-foreground: oklch(0.145 0 0);
  --primary: #030213;
  --primary-foreground: oklch(1 0 0);
  --secondary: oklch(0.95 0.0058 264.53);
  --secondary-foreground: #030213;
  --muted: #ececf0;
  --muted-foreground: #717182;
  --accent: #e9ebef;
  --accent-foreground: #030213;
  --destructive: #d4183d;
  --destructive-foreground: #ffffff;
  --border: rgba(0, 0, 0, 0.1);
  --input: transparent;
  --input-background: #f3f3f5;
  --switch-background: #cbced4;
  --font-weight-medium: 500;
  --font-weight-normal: 400;
  --ring: oklch(0.708 0 0);
  --chart-1: oklch(0.646 0.222 41.116);
  --chart-2: oklch(0.6 0.118 184.704);
  --chart-3: oklch(0.398 0.07 227.392);
  --chart-4: oklch(0.828 0.189 84.429);
  --chart-5: oklch(0.769 0.188 70.08);
  --radius: 0.625rem;
  --sidebar: oklch(0.985 0 0);
  --sidebar-foreground: oklch(0.145 0 0);
  --sidebar-primary: #030213;
  --sidebar-primary-foreground: oklch(0.985 0 0);
  --sidebar-accent: oklch(0.97 0 0);
  --sidebar-accent-foreground: oklch(0.205 0 0);
  --sidebar-border: oklch(0.922 0 0);
  --sidebar-ring: oklch(0.708 0 0);
}

.dark {
  --background: oklch(0.145 0 0);
  --foreground: oklch(0.985 0 0);
  --card: oklch(0.145 0 0);
  --card-foreground: oklch(0.985 0 0);
  --popover: oklch(0.145 0 0);
  --popover-foreground: oklch(0.985 0 0);
```

```
  --primary: oklch(0.985 0 0);
  --primary-foreground: oklch(0.205 0 0);
  --secondary: oklch(0.269 0 0);
  --secondary-foreground: oklch(0.985 0 0);
  --muted: oklch(0.269 0 0);
  --muted-foreground: oklch(0.708 0 0);
  --accent: oklch(0.269 0 0);
  --accent-foreground: oklch(0.985 0 0);
  --destructive: oklch(0.396 0.141 25.723);
  --destructive-foreground: oklch(0.637 0.237 25.331);
  --border: oklch(0.269 0 0);
  --input: oklch(0.269 0 0);
  --ring: oklch(0.439 0 0);
  --font-weight-medium: 500;
  --font-weight-normal: 400;
  --chart-1: oklch(0.488 0.243 264.376);
  --chart-2: oklch(0.696 0.17 162.48);
  --chart-3: oklch(0.769 0.188 70.08);
  --chart-4: oklch(0.627 0.265 303.9);
  --chart-5: oklch(0.645 0.246 16.439);
  --sidebar: oklch(0.205 0 0);
  --sidebar-foreground: oklch(0.985 0 0);
  --sidebar-primary: oklch(0.488 0.243 264.376);
  --sidebar-primary-foreground: oklch(0.985 0 0);
  --sidebar-accent: oklch(0.269 0 0);
  --sidebar-accent-foreground: oklch(0.985 0 0);
  --sidebar-border: oklch(0.269 0 0);
  --sidebar-ring: oklch(0.439 0 0);
}

@theme inline {
  --color-background: var(--background);
  --color-foreground: var(--foreground);
  --color-card: var(--card);
  --color-card-foreground: var(--card-foreground);
  --color-popover: var(--popover);
  --color-popover-foreground: var(--popover-foreground);
  --color-primary: var(--primary);
  --color-primary-foreground: var(--primary-foreground);
  --color-secondary: var(--secondary);
  --color-secondary-foreground: var(--secondary-foreground);
  --color-muted: var(--muted);
  --color-muted-foreground: var(--muted-foreground);
  --color-accent: var(--accent);
  --color-accent-foreground: var(--accent-foreground);
  --color-destructive: var(--destructive);
  --color-destructive-foreground: var(--destructive-foreground);
  --color-border: var(--border);
  --color-input: var(--input);
  --color-input-background: var(--input-background);
  --color-switch-background: var(--switch-background);
  --color-ring: var(--ring);
```

```
  --color-chart-1: var(--chart-1);
  --color-chart-2: var(--chart-2);
  --color-chart-3: var(--chart-3);
  --color-chart-4: var(--chart-4);
  --color-chart-5: var(--chart-5);
  --radius-sm: calc(var(--radius) - 4px);
  --radius-md: calc(var(--radius) - 2px);
  --radius-lg: var(--radius);
  --radius-xl: calc(var(--radius) + 4px);
  --color-sidebar: var(--sidebar);
  --color-sidebar-foreground: var(--sidebar-foreground);
  --color-sidebar-primary: var(--sidebar-primary);
  --color-sidebar-primary-foreground: var(--sidebar-primary-foreground);
  --color-sidebar-accent: var(--sidebar-accent);
  --color-sidebar-accent-foreground: var(--sidebar-accent-foreground);
  --color-sidebar-border: var(--sidebar-border);
  --color-sidebar-ring: var(--sidebar-ring);
}

@layer base {
  * {
    @apply border-border outline-ring/50;
  }

  body {
    @apply bg-background text-foreground;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
  }
}

/**
 * Base typography. This is not applied to elements which have an ancestor with a Tailwind
text class.
 */
@layer base {
  :where(:not(:has([class*=' text-']), :not(:has([class^='text-'])))) {
    h1 {
      font-size: var(--text-2xl);
      font-weight: var(--font-weight-medium);
      line-height: 1.5;
    }

    h2 {
      font-size: var(--text-xl);
      font-weight: var(--font-weight-medium);
      line-height: 1.5;
    }

    h3 {
      font-size: var(--text-lg);
      font-weight: var(--font-weight-medium);
```

```
      line-height: 1.5;
    }

    h4 {
      font-size: var(--text-base);
      font-weight: var(--font-weight-medium);
      line-height: 1.5;
    }

    p {
      font-size: var(--text-base);
      font-weight: var(--font-weight-normal);
      line-height: 1.5;
    }

    label {
      font-size: var(--text-base);
      font-weight: var(--font-weight-medium);
      line-height: 1.5;
    }

    button {
      font-size: var(--text-base);
      font-weight: var(--font-weight-medium);
      line-height: 1.5;
    }

    input {
      font-size: var(--text-base);
      font-weight: var(--font-weight-normal);
      line-height: 1.5;
    }
  }
}

html {
  font-size: var(--font-size);
}
```