Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота
за освітнім ступенем «бакалавр»
на тему:

ПОРІВНЯЛЬНИЙ АНАЛІЗ БІБЛІОТЕК ДЛЯ ВІЗУАЛІЗАЦІЇ ДАНИХ НА JAVA SCRIPT

Виконав:
здобувач IV курсу
групи ІПЗ-41
спеціальності 121 «Інженерія програмного забезпечення»
Тимощук Олександр Станіславович

Науковий керівник:
к.т.н., доцент Батишкіна Ю. В.

Рівне – 2024

АНТОТАЦІЯ

Тимощук О. С. «Порівняльний аналіз бібліотек для візуалізації даних на JavaScript». – Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 121 Інженерія програмного забезпечення – Рівненський державний гуманітарний університет – Рівне, 2024. – 67 с.

У роботі розглянуті основні можливості бібліотек JavaScript для візуалізації даних та проведений їх порівняльний аналіз. Візуалізація даних виступає критично важливим інструментом для аналізу та інтерпретації великих масивів інформації.

Проведений теоретичний аналіз засвідчив, що для створення складних та гнучких візуалізацій рекомендується D3.js, для простих та швидких – Chart.js або Google Charts, остання з яких не потребує знання JavaScript. Вибір бібліотеки залежить від конкретних потреб проекту та досвіду розробника.

Для порівняльної характеристики бібліотек для візуалізації даних визначено шість основних критеріїв, як-от: інтуїтивність інтерфейсу, доступність документації та навчальних матеріалів, наявність інструментів та засобів для відлагодження.

Розроблений план тестування візуалізації створених за допомогою бібліотек D3.js, Chart.js та Google Charts. Тестування проводилося на функціональність, ергономічність, простота, продуктивність, швидкодія, сумісність, адаптивність, надійність.

Встановлено, що найбільш оптимальною за даними критеріями, усереднено можна вважати бібліотеку Google Charts. Водночас представлені результати тестування та порівняльного аналізу бібліотек візуалізації даних D3.js, Chart.js та Google Charts дозволяють зробити обґрунтований вибір оптимальної бібліотеки для конкретного проекту. На результати могли суб’єктивні технічні фактори (стабільність Інтернет підключення та роботи ПК) й судження експертів.

Подальші дослідження вбачаємо у розробці автоматизованих тестів порівняння бібліотек Java Script для візуалізації даних.

Ключові слова: візуалізація даних, бібліотеки Java Script, тестування якості, порівняльне тестування, D3.JS, Chart.JS, Google Charts.

ЗМІСТ

ВСТУП

Візуалізація даних виступає критично важливим інструментом для аналізу та інтерпретації великих масивів інформації. Вона дозволяє презентувати дані у зрозумілому та інтерактивному форматі, що сприяє оптимізації процесу прийняття техніко-економічних рішень. Наразі існує велика кількість технологій для візуалізації даних, які забезпечують вирішення різноманітних технічних завдань. Зокрема, веб-технології займають провідну позицію, що робить їх особливо актуальними для візуалізації даних з веб-застосунків. Важливо відзначити, що бібліотеки JavaScript для візуалізації даних надають широкий спектр функціональних можливостей, які дозволяють створювати візуалізації різного рівня складності та призначення. Вибір відповідної бібліотеки залежить від специфічних вимог та цілей проекту.

Порівняльний аналіз бібліотек JavaScript для візуалізації даних є важливим етапом у процесі розробки програмного забезпечення, оскільки він дозволяє розробникам здійснити надійний вибір оптимальної бібліотеки для конкретного проекту. Такий аналіз включає оцінку продуктивності, функціональних можливостей, масштабованості, сумісності з іншими технологіями, а також підтримки та документованості бібліотек. Враховуючи ці критерії, розробники можуть визначити, яка бібліотека найкраще відповідає специфічним вимогам проекту, забезпечуючи ефективне та надійне представлення даних.

Враховуючи актуальність цієї проблематики та недостатню її розробленість у теорії та практиці комп'ютерних технологій й розробки програмного забезпечення, зокрема відсутність комплексного порівняльного аналізу їхніх функціональних можливостей і недостатню розробленість інструментів для тестування й порівняння їх техніко-функціональних характеристик, було обрано тему нашого дослідження (кваліфікаційної роботи): **«Порівняльний аналіз бібліотек для візуалізації даних на JavaScript».**

Метою дослідження є теоретичне порівняння бібліотек для візуалізації на JavaScript та розробка інструментів для їхнього тестування. Досягнення цієї мети вимагає виконання таких завдань:

1. Провести детальний огляд наукової літератури та технічної документації.

2. Виконати теоретико-порівняльний аналіз характеристик бібліотек для візуалізації даних на JavaScript.

3. Розробити детальний план тестування бібліотек для візуалізації даних на JavaScript.

4. Створити модулі для тестування, призначені для практичного дослідження характеристик бібліотек для візуалізації даних на JavaScript.

5. Експериментально порівняти бібліотеки шляхом їхнього тестування.

Об'єкт дослідження – бібліотеки JavaScript для візуалізації даних.

Предмет дослідження – порівняльний аналіз бібліотек JavaScript для візуалізації даних.

Для вирішення поставлених завдань були використані наступні методи дослідження:

- *Теоретичні методи*: теоретичний аналіз технічної літератури, наукових статей та офіційної документації з теми дослідження; систематизація та порівняння.
- *Емпіричні методи*: проведення порівняльного аналізу бібліотек за наступними критеріями:
  - типи підтримуваних візуалізацій;
  - джерела даних, що можуть використовуватися;
  - можливості інтерактивності та адаптивності;
  - складність використання;
  - доступність бібліотек.
- *Статистичні методи*: застосування статистичних методів для обробки та аналізу даних, отриманих в ході емпіричних досліджень.

Методи дослідження обрані в залежності від завдань дослідження та конкретних обставин.

Структура роботи. Робота охоплює 4 розділи, 10 додатків. Кількість найменувань у списку використаних джерел складає 22 одиниці. Кількість

таблиць – 1, кількість рисунків – 17. Обсяг роботи до висновків включно, без врахування списку використаних джерел і додатків становить 45 ст., загальний обсяг – 67 ст.

РОЗДІЛ І. ТЕОРЕТИЧНІ ОСНОВИ ВІЗУАЛІЗАЦІЇ ДАНИХ

1.1. Основи візуалізації даних

Технологія візуалізації даних відіграє ключову роль у трансформації необроблених даних у графічні зображення, забезпечуючи ефективне унаочнення інформації. Вона має широке застосування в прийнятті бізнес-рішень, як-от прогнозування ринкового розвитку, виявлення проблем, аналіз купівельних тенденцій, точний аналіз даних та дослідження прихованих закономірностей. Трансформування даних у візуальний формат за допомогою точного програмного забезпечення є основною потребою сучасного бізнесу, науки й виробництва. Ефективні інструменти візуалізації є невід'ємною складовою цієї технології. На практиці використовуються різноманітні інструменти візуалізації даних, що охоплюють карти, графіки, діаграми, схеми та інфографіку. Вибір оптимального інструменту та техніки візуалізації є основним професійним викликом для осіб, які приймають рішення. Для цього необхідні знання різних інструментів візуалізації та розуміння їх методів, що є важливою навичкою для майбутніх розробників програмного забезпечення.

Порівняльне дослідження інструментів візуалізації даних обґрунтовується на форматі вихідних даних, потреби в знаннях кодування та комерційній частині інструменту. Порівняльне дослідження методів візуалізації даних за допомогою діаграми представляє деякі розширені типи діаграм разом з їх відповідними функціями та логічним дизайном [1, с. 48].

Головною філософією візуалізації даних є «достатність». Едвард Бафт(1983) — піонер у галузі компютерної візуалізації даних, зауважує що необхідно говорити правду та «Візуалізувати якомога більше даних» [23].

Як відомо, "зображення варте тисячі слів" — а часто й більше, — проте це справедливо лише тоді, коли графічне представлення є найбільш підходящим способом передачі інформації, а саме зображення є ретельно оформленим. Можна тривалий час аналізувати таблицю числових даних і не помітити того, що стане

очевидним при погляді на якісну візуалізацію тих самих даних. Як приклад ми маємо ряд випадкових даних (рис.1), які представлені у вигляді таблиці:

Region	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total
Domestic	1,983	2,343	2,593	2,283	2,574	2,838	2,382	2,634	2,938	2,739	2,983	3,493	31,783
International	574	636	673	593	644	679	593	139	599	583	602	690	7,005
Total	2,557	2,979	3,266	2,876	3,218	3,517	2,975	2,773	3,537	3,322	3,585	4,183	38,788

Рис. 1.1 Табличні дані про результати продажів по двох регіонах

Ця таблиця виконує дві важливі функції: вона достовірно відображає значення продажів і забезпечує ефективний підхід для пошуку даних за конкретним регіоном та місяцем. Однак, якщо наше завдання полягає у виявленні певних тенденцій або аномалій серед цих значень, у швидкому сприйнятті інформації, закладеної в цих числах, або у порівнянні цілих наборів даних, а не лише двох значень одночасно, ця таблиця репрезентує недостатню ефективність.

S. Few пропонує використовувати засоби візуалізації в залежно від потреб сприйняття. Зокрема у нашому випадку його підхід ґрунтується на подані інформації без «вербальної обробки» [3]. Він пропонує представляти такі дані у вигляді діаграм, які дозволяють прослідкувати певну динаміку й тренди (рис. 1.2.).

Згідно з його підходом можна виокремити кілька закономірностей:

1. Внутрішні продажі були значно вищими та стабільно перевершували міжнародні продажі.
2. Внутрішні продажі демонстрували загальну тенденцію до зростання протягом року.
3. Міжнародні продажі залишалися відносно стабільними, за винятком серпня, коли вони різко скоротилися.
4. Внутрішні продажі мали циклічний характер: вони зростали протягом двох місяців, а потім знижувалися, досягаючи піку в останній місяць кварталу і різко падаючи в першому місяці наступного кварталу.

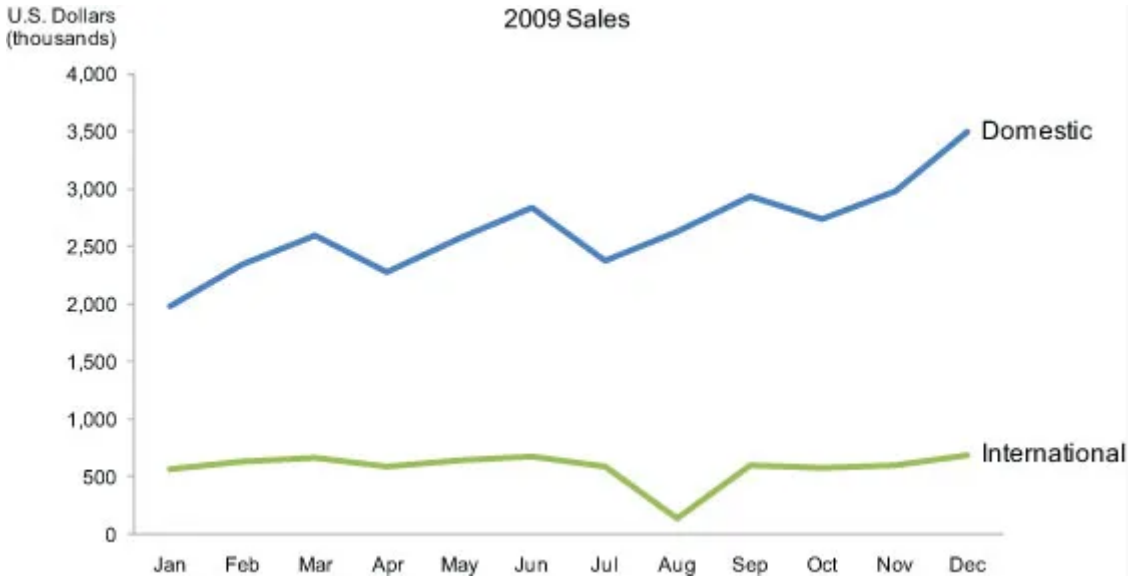


Рис. 1.2. Візуальне представлення табличних даних (підхід S. Few).

Тобто головним у візуалізації є наступне (у порядку пріоритетності): зображення необхідних даних, чітке розуміння та усвідомлення, повнота даних (без перевантаження), можливість встановлювати тренди, можливість визначення кореляції між досліджуваними явищами.

Водночас необхідно розуміти, що окреслені пріоритети візуалізації даних не є дієвими для різних типів візуалізації. Більш особливими у контексті представлення даних є: ретроспективні візуалізації, наочності про людське сприйняття, кількісно-статистичні візуалізації з багатофакторними похідними.

Візуалізація даних виконує три основні функції: дослідження, підтвердження та презентація, які можуть бути подані у виді різних типів графічних представлень. Аналіз видів візуалізації буде виконано у наступному параграфі.

1.2. Основні типи графічних представлень візуалізації даних

У сучасній практиці візуалізації даних утворилися негласні (не стандартизовані) правила представлення даних у вигляді графічних об’єктів. Переважна більшість дослідників використовує термін діаграма, що означає

«графічний опис даних і простий спосіб відображення інформації» [4]. До класичних прикладів візуалізації належать такі:

Стовпчасті діаграми (або гістограми).

Переважно використовуються для порівняння даних між категоріями або для того, щоб показати як досліджуваний об’єкт або явище змінюється у ретроспективі [5]. Стовпчасті діаграми (Рис. 1.4.) та горизонтальні види є найбільш популярними, але можуть зустрічатися комбіновані, тривимірні, багаторядні й з накопиченням.

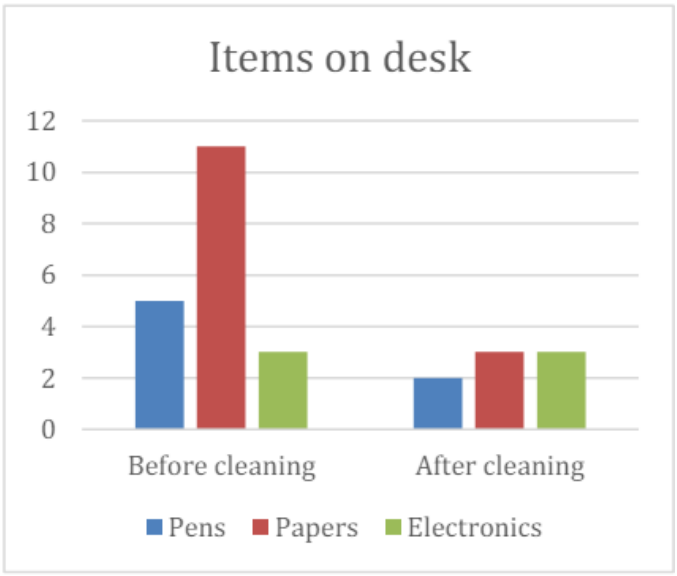


Рис. 1.3. Приклад вертикальної гістограми

Гістограми – це популярний і лаконічний тип візуалізації даних, що використовується для чіткої демонстрації розподілу даних. Ними послуговуються у різних галузях, таких як наука, бізнес і соціальні науки, для візуалізації даних про все, від експресії генів до цін на акції й успішності здобувачів освіти. Завдяки своїй простоті та універсальності гістограми роблять цінним інструментом для дослідників, аналітиків та будь-кого, хто прагне зрозуміти данні.

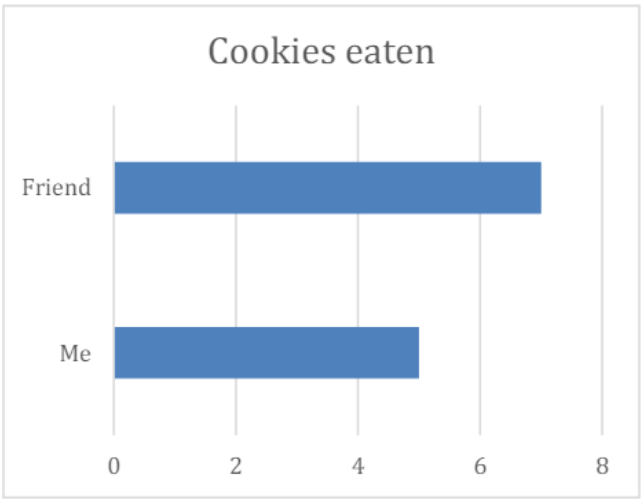


Рис. 1.4. Приклад горизонтальної гістограми

Лінійна діаграма (графік).

Використовуються переважно для унаочнення тенденцій даних в ретроспективі, дозволяють порівняти декілька наборів даних, а також візуалізувати зв'язок між двома змінними (рис. 1.5).

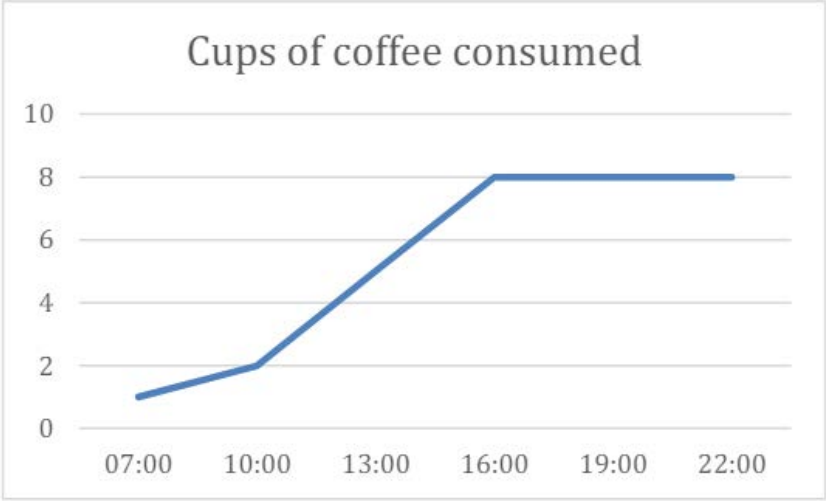


Рис. 1.5. Приклад лінійної діаграми

Бульбашкова діаграма.

Цією діаграмою варто скористатися, якщо необхідно потрібно змінити масштаб певної осі або потрібно показати цю вісь у логарифмічному масштабі. Також її доцільно використати коли є необхідність продемонструвати схожість між великими наборами даних, а не різницю між точками даних [6, с.28].

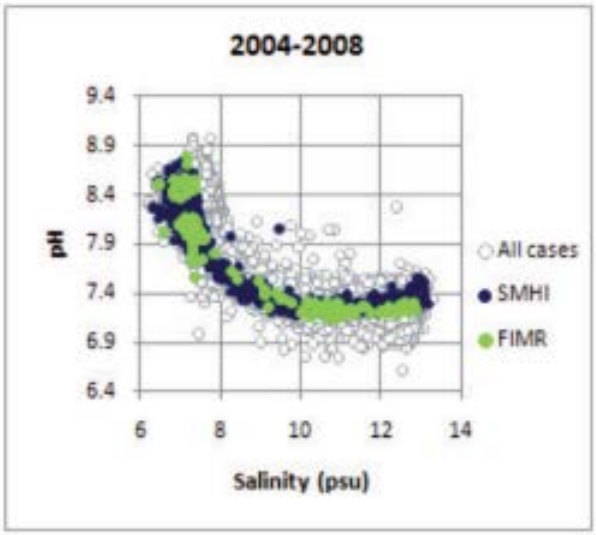


Рис. 1.6. Приклад бульбашкової діаграми

Секторна діаграма.

Такий тип доцільно використати для унаочнення ієрархічних даних і може бути представлена, коли необхідно визначити частки за певними змінними (рис. 1.7).

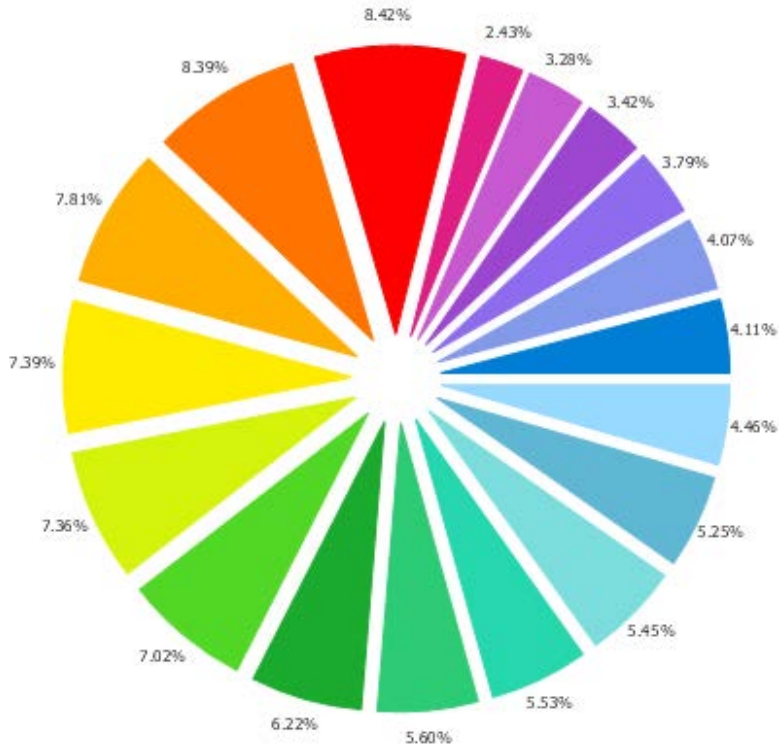


Рис. 1.7. Приклад секторної діаграми

Існує велика кількість діаграм однак найбільш популярними та апробованими у мовах сучасності є саме ці. Порівняння ефективності бібліотек візуалізації Javascript буде здійснюватися з використанням описаних у цьому параграфі типів.

1.3. Передумови, методи та засоби візуалізації даних веб застосунків та веб сайтів

Візуалізація даних – це ефективний метод, який допомагає дослідникам та аналітикам отримувати глибокі знання з великих обсягів інформації. Її ефективність ґрунтується на здатності людського мозку обробляти та інтерпретувати візуальну інформацію з більшою швидкістю та легкістю, ніж текстову [6]. Завдяки візуалізації дані трансформуються в зображення, які користувачі можуть сприймати та аналізувати, виконуючи три основні підзадачі:

1. **Перцептивне групування:** Об'єднання елементів даних, які мають схожі характеристики, візуально, щоб полегшити їхнє розуміння.
2. **Сегментація зображення:** Розбиття зображення на окремі, чітко визначені сегменти, що відповідають різним категоріям або значенням даних.
3. **Розпізнавання об'єктів:** Ідентифікація та класифікація окремих об'єктів або елементів у візуалізації даних [7].

Перед тим, як розпочати візуалізацію даних, необхідно чітко визначити дві ключові характеристики: цільову аудиторію та тип користувача. Ці фактори значно впливають на вибір методів та інструментів візуалізації, а також на структуру презентації даних.

1. **Визначення цільової аудиторії:** Розуміння того, кому призначена візуалізація даних, є важливим кроком для забезпечення її ефективності. Знання рівня підготовки, досвіду та очікувань аудиторії дозволяє візуалізувати дані в спосіб, який буде їм зрозумілим та цікавим.
2. **Ідентифікація типу користувача:** Необхідно також враховувати тип користувача, який буде взаємодіяти з візуалізацією даних. Це може бути

пасивний користувач, який просто переглядає інформацію, або активний користувач, який буде досліджувати та аналізувати дані. Тип користувача впливає на те, скільки інформації слід представити, а також на рівень інтерактивності візуалізації.

3. **Врахування часу, доступного користувачу:** Важливо знати, скільки часу користувач готовий витратити на вивчення візуалізації даних. Це може вплинути на вибір складності візуалізації та на те, скільки інформації слід представити одночасно.

Методи візуалізації даних для різних типів користувачів:

Існує багато різних методів та інструментів візуалізації даних, кожен з яких має свої переваги та недоліки. Найкращий метод для конкретної ситуації залежить від цільової аудиторії, типу користувача та доступного часу.

- **Пасивні користувачі:** Для пасивних користувачів, які просто переглядають інформацію, можуть підходити прості та чіткі візуалізації, такі як діаграми або графіки.
- **Активні користувачі:** Для активних користувачів, які досліджують та аналізують дані, можуть бути більш доречними інтерактивні візуалізації, такі як карти даних або панелі інструментів.
- **Обмежений час:** Якщо у користувача мало часу, може бути доцільним використовувати візуалізацію, яка швидко дає огляд даних, наприклад, кругову діаграму або гістограму.

Класифікація методів візуалізації даних:

Методи візуалізації даних можна класифікувати на три основні категорії:

1. **Описова візуалізація:** Цей тип візуалізації використовується для представлення даних у простій та зрозумілій формі. До прикладів описової візуалізації належать діаграми, графіки та карти.
2. **Аналітична візуалізація:** Цей тип візуалізації використовується для допомоги користувачам у виявленні закономірностей та тенденцій у даних. До прикладів аналітичної візуалізації належать теплові карти, дерева рішень та мережеві графіки.

3. **Інтерактивна візуалізація:** Цей тип візуалізації дозволяє користувачам досліджувати та аналізувати дані в інтерактивному режимі. До прикладів інтерактивної візуалізації належать карти даних, панелі інструментів та візуалізації на основі часу.

Вибір методів та інструментів візуалізації даних є важливим кроком для забезпечення ефективної комунікації інформації. Розуміння цільової аудиторії, типу користувача та доступного часу дозволяє візуалізувати дані в спосіб, який буде зрозумілим, цікавим та корисним.

Зазначимо, що за кілька десятиліть існування Інтернету відбулися значні зміни, і темпи цих змін настільки стрімкі, що їх важко наздогнати. Технології продовжують розвиватися щосекунди, породжуючи нові тенденції, які з'являються і зникають з такою ж швидкістю. На сьогоднішній день швидкість завантаження веб-сайту є одним з найважливіших факторів, що впливають на якість користувацького досвіду. Час завантаження веб-сторінки визначає, чи будуть відвідувачі продовжувати користуватися сайтом, рекомендувати його іншим та повертатися знову. Дослідження показують, що якщо веб-сайт завантажується більше двох секунд, понад половина відвідувачів схильна покинути його. Надмірна кількість HTTP-запитів, таких як завантаження таблиць стилів CSS, скриптів, HTML-документів, зображень тощо, призводить до збільшення часу рендерингу, що уповільнює роботу веб-сайту.

Візуалізація даних має забезпечувати прямий, точний і оперативний доступ до інформації для своєї аудиторії, забезпечуючи швидкий доступ до будь-яких даних, представлених у діаграмі. Дані повинні бути доступні на відстані одного кліку. Однак, зі збільшенням обсягу даних точок, час завантаження та відображення сторінки збільшується. Одним із факторів, який обмежує продуктивність при роботі з великими обсягами веб-даних, є рендеринг DOM, особливо при використанні ванільного JavaScript, що може призводити до повільної роботи DOM.

З періоду інтенсивного розвитку соціальних мереж, таких як Facebook, Instagram, Twitter та інші, обсяг контенту зросло, що спричинило очевидне

сповільнення завантаження веб-ресурсів [9]. У порівнянні з минулим, коли веб-сторінки були простішими і взаємодія з ними не мала такого значення, як сьогодні, взаємодія людини з комп'ютером (Human-Computer Interaction, HCI) зараз вимагає, щоб власники веб-сайтів забезпечували швидкість завантаження сторінок менше ніж за 2 секунди. Якщо час завантаження сторінки перевищує 2 секунди, це може негативно позначитися на конверсійному маркетингу. Лідери у сфері технологій, такі як Google, Mozilla, Shopzilla та Netflix [8], визнали, що продуктивність є критично важливою характеристикою Інтернету, яка впливає на рівень доходів та їх коливання [10].

Для наших задач дуже ефективними є бібліотеки JavaScript, які відіграють важливу роль у веб-розробці, надаючи розробникам доступ до готових компонентів та функцій, які можна повторно використовувати. Це сприяє економії часу та ресурсів, а також забезпечує узгодженість та стандартизацію коду.

Переважна більшість бібліотек мають відкритий код – означає, що вихідний код програмного забезпечення доступний публічно та безкоштовно для використання, модифікації та поширення. Це дає багатьом розробникам можливість вносити свій вклад у вдосконалення та розширення функціональності бібліотеки.

Разом з тим, досить ефективними є бібліотеки та засоби на основі мов програмування C# та Python, однак їх використання вимагає інтегрування веб-сервера, що вимагає додаткових налаштувань. Крім того їхню адаптацію до мови розмітки веб сторінки та каскадних таблиць стилей виконувати досить складно, що потребує додаткових інтелектуальних й часових ресурсів. Відтак в якості засобів розробки візуалізації даних доцільно використовувати саме бібліотеки Java Script.

Висновки до розділу 1.

Візуалізація даних є невід'ємною частиною сучасного наукового та аналітичного процесу. Використання ефективних методів візуалізації та відповідних інструментів дозволяє дослідникам, аналітикам та фахівцям з даних отримувати глибокі знання з великих обсягів інформації, а також ефективно презентувати результати своєї роботи широкій аудиторії. На сьогоднішній день найефективнішим підходом для візуалізації веб застосунків і веб сайтів є бібліотеки Java Script.

РОЗДІЛ 2. ТЕОРЕТИЧНИЙ АНАЛІЗ ХАРАКТЕРИСТИК БІБЛІОТЕК ДЛЯ ВІЗУАЛІЗАЦІЇ ДАНИХ НА JAVASCRIPT

2.1. Характеристика бібліотеки D3.js

D3.js – це популярна бібліотека JavaScript для візуалізації даних. Вона пропонує широкий спектр функцій та можливостей, що дозволяють створювати візуалізації різної складності та призначення (рис.2.1).

Рис. 2.1. Логотип бібліотеки D3.js

D3.js (або просто D3 для документів, керованих даними) – це бібліотека JavaScript для створення динамічних, інтерактивних візуалізацій даних у веб-браузерах. Він використовує широко впроваджені стандарти SVG, HTML5 і CSS. Вона є наступником попередньої системи Protovis. На відміну від багатьох інших бібліотек, D3.js забезпечує великий контроль над кінцевим візуальним результатом (рис.2.2). Були різні попередні спроби візуалізації даних до веб-браузерів. Бібліотека JavaScript D3.js, вбудована у веб-сторінку HTML, використовує попередньо побудовані функції JavaScript для вибору елементів, створення об’єктів SVG, їх стилізування або додавання до них переходів, динамічних ефектів або підказок. Ці об’єкти також можна широко використовувати за допомогою CSS [11].

Рис. 2.2. Можливості візуалізації бібліотеки D3.js

D3.js має у своєму арсеналі низку методів для роботи з даними. Згідно із довідковими сторінками API бібліотек на gitHub [14] ця бібліотека має значну кількість візуалізаційних засобів, зокрема детальніше про них.

Ядро – це місце, де дані розміщуються у вибрані DOM-елементи, де створюються переходи, завантажуються дані, додаються кольори до візуалізації та всі стандартні методи, що використовуються для перенесення даних на HTML-сторінку.

Географія, як випливає з назви, є частиною картографічних візуалізацій.

Геометрія дозволяє створювати макети «воронок», чотирикутників, полігонів і корпусів. Макети можуть створювати кілька стандартних графіків без особливого кодування з боку програміста.

Також існує стандартний макет гістограми, пакування (рекурсивне пакування колами), розбиття (дерево вузлів на сонячні промені або бурульки), традиційний пиріг), традиційна кругова діаграма, стек, дерево і карта дерева (відображення дерева вузлів) [12, 11].

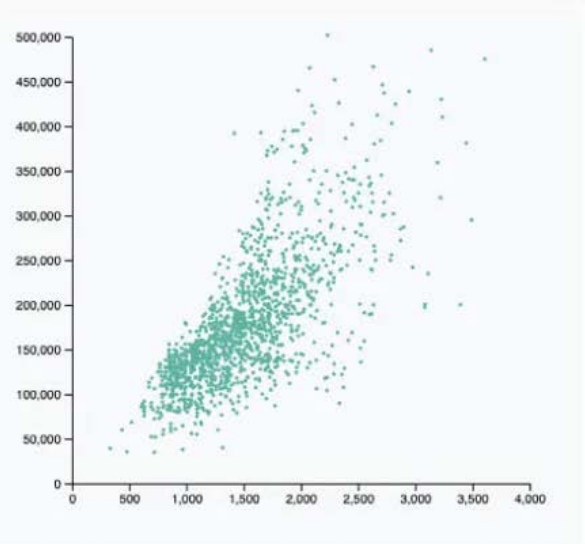
Масштаби дозволяють розробнику працювати з неперервними вхідними доменами, такими як числа, дискретними вхідними доменами, такими як імена або дискретними доменами вводу, такими як імена, категорії або часові домени. SVG прив'язує візуальні елементи до даних і DOM. Метод часу має окрему групу, оскільки дозволяє, наприклад, конвертувати часовий формат і розміщувати інтервали.

Крім того D3.js працює з форматом SVG – це векторним форматом, який був стандартизований для Інтернету і працює у всіх основних браузерах. Він описується досить простим синтаксисом, який дуже схожий на HTML [13]. Він містить безліч властивостей стилів, які можна використовувати для створення графіки. Деякі з них – це колір і прозорість (альфа-канал), властивості обведення, порядок накладання, а також групи і трансформації.

Для завантаження бібліотеки достатньо прописати один рядок:

```
<script src="//d3js.org/d3.v7.min.js"></script>
```

Розширення функціоналу діаграм за допомогою вбудованих трансформацій D3 та створення індивідуальних візуалізацій пов'язане з низкою складнощів. Зокрема, для побудови базової діаграми розсіювання користувачу необхідно виконати ряд послідовних кроків: збір та обробку даних, створення відповідних осей та області діаграми, візуалізацію та розміщення елементів SVG у DOM-структурі.




```
// Add Y axis
var y = d3.scaleLinear()
  .domain([0, 500000])
  .range([ height, 0]);
svg.append("g")
  .call(d3.axisLeft(y));

// Add dots
svg.append('g')
  .selectAll("dot")
  .data(data)
  .enter()
  .append("circle")
    .attr("cx", function (d) { return x(d.GrLivArea); } )
    .attr("cy", function (d) { return y(d.SalePrice); } )
    .attr("r", 1.5)
    .style("fill", "#69b3a2")
```

Рис. 2.3. Приклад діаграми D3 і фрагмент коду

Значний потенціал налаштування робить цю бібліотеку не тільки складною у користуванні, а й не обмежує її функціоналу. До прикладу її використовують як основну такі бренди як Google News, The New York Times, The Upshot та інші.

2.2. Характеристика бібліотеки Chart.js

Chart.js – це безкоштовна бібліотека JavaScript, що використовується для візуалізації даних, з використанням полотна HTML5 і є однією із найпопулярніших на Github.

Вказана бібліотека використовується для генерації у клієнта (браузера комп'ютера) графіків і діаграм. Для цього використовуються стандартні засоби для роботи з клієнтською частиною веб-додатка – HTML, CSS, JS.

Рис. 2.4. Логотип Chart.js

Переваги в такому:

- Доступність. Ця бібліотека є opensource-проектом, у зв'язку з цим нею можна користуватися безкоштовно.

- Проста адаптивність. Ядро бібліотеки дає змогу адаптувати графіки під смартфони, планшети, комп'ютери

- Невеликий розмір бібліотеки. Завдяки цьому завантаження графіків проходить досить швидко.

- Підтримка анімацій.

- Велика кількість опцій для налаштування графіків.

Таким чином, ChartJS є одним з ефективних інструментів для візуалізації даних. Візуалізація даних у сфері Вебу може відрізнятися, оскільки зараз веб-додатки можуть бути розроблені для різних сфер, від простих інтернет-магазинів до відображення сигналів датчиків [15].

Наведемо приклад візуалізації (Рис. 2.5).

Згідно з рисунком помітно, що усі дані зберігаються у колекціях. В цьому конкретному випадку це колекції labels і database. Масив data виступає у ролі точок, які формують графік. Тобто дані підставлені у масив браузер може візуалізувати у вигляді діаграми. Слід відмітити, що синтаксис візуалізації має досить зрозумілу та лаконічну форму, що робить ChartJS досить простою у використанні.

```
const config = {
  type: 'polarArea',
  data: data,
  options: {
    responsive: true,
    plugins: {
      legend: {
        position: 'top',
      },
      title: {
        display: true,
        text: 'Chart.js Polar Area Chart'
      }
    }
  },
};
```

```
const DATA_COUNT = 5;
const NUMBER_CFG = {count: DATA_COUNT, min: 0, max: 100};

const labels = ['Red', 'Orange', 'Yellow', 'Green', 'Blue'];
const data = {
  labels: labels,
  datasets: [
    {
      label: 'Dataset 1',
      data: Utils.numbers(NUMBER_CFG),
      backgroundColor: [
        Utils.transparentize(Utils.CHART_COLORS.red, 0.5),
        Utils.transparentize(Utils.CHART_COLORS.orange, 0.5),
        Utils.transparentize(Utils.CHART_COLORS.yellow, 0.5),
        Utils.transparentize(Utils.CHART_COLORS.green, 0.5),
        Utils.transparentize(Utils.CHART_COLORS.blue, 0.5),
```

Рис. 2.5. Синтаксис візуалізації засобами ChartJS

Тобто таким чином ми можемо представити дані, котрі приходять із сервера. Це можуть бути результати розрахунків або сигнали котрі отримуємо від певного обладнання (рис. 3).

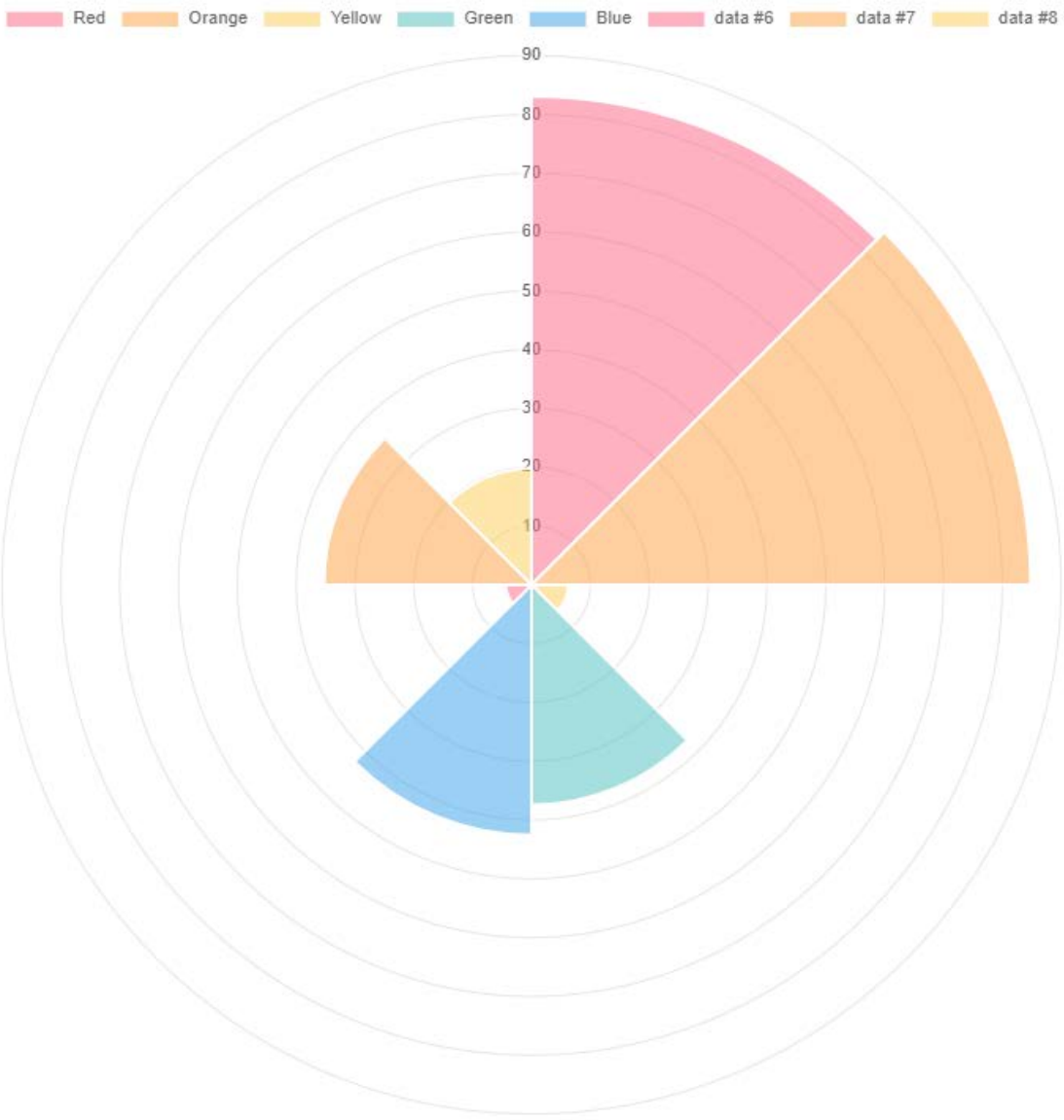


Рис. 2.6. Приклад візуалізації даних за допомогою ChartJS

У наш час бібліотека ChartJS представляє собою вдосконалений інструмент для візуалізації даних, репрезентуючи неабияку ефективність у вирішенні завдань з аналізу та представлення інформації. Її надзвичайна привабливість полягає у тому, що вона є повністю безкоштовною і вирізняється високим рівнем доступності для користувачів з різним рівнем навичок.

На сьогоднішній день ChartJS займає заслужене місце серед найпопулярніших бібліотек у Інтернет-спільноті, що свідчить про її великий запит та довіру спільноти розробників. Велика кількість проєктів та веб-додатків

активно використовують цей інструмент завдяки його гнучкості та можливостям кастомізації.

Для встановлення бібліотеки необхідно прописати:

`npm install chart,js`, або `yarn add chart.js`

Наголос треба робити на тому, що функціональні можливості бібліотеки ChartJS не лише відповідають сучасним вимогам, а й постійно розширюються. Це підтверджується регулярними оновленнями та додатковими розробками, які роблять цей інструмент вкрай актуальним для широкого спектра завдань в області візуалізації даних на платформі JavaScript.

Для якісної інтеграції візуалізації створеної за допомогою цієї бібліотеки потрібно чітко розуміти базові технології веб-розробки (HTML, CSS), а також спеціальні інструменти (JQuery fundamentals).

В якості масивів даних бібліотека ChartJS використовує звичні формати: CSV, XML, JSON.

Незважаючи на простоту базових функцій, Chart.js може здатися складним інструментом через велику кількість параметрів та налаштувань. У процесі аналізу цієї бібліотеки ми значно розширили свої знання щодо візуалізації даних та методів ефективного використання елемента canvas HTML5. Загалом, Chart.js слід розглядати як цінний інструмент для створення візуальних уявлень, що сприяють кращому розумінню та аналізу даних.

Бібліотека пропонує широкий спектр методів для побудови діаграм. Користувач може вибрати простий підхід для візуалізації одного набору даних, створювати динамічні діаграми, що відображають динаміку декількох наборів даних, або розробляти складні діаграми з даними, імпортованими з API.

2.3. Характеристики бібліотеки Google Charts.

Google Charts – це бібліотека JavaScript для візуалізації даних, яка пропонує широкий спектр функцій та можливостей. Вона є популярною вибором для розробників, які хочуть створювати візуалізації даних для веб-сайтів і додатків.

Рис. 2.7. Логотип Google Charts

Функціонал Google Charts Tools включає в себе:

- динамічні піктограми;
- карти;
- циферблати і дисплеї;
- формули;
- можливість створювати свої інструменти візуалізації і використовувати сторонні ресурси.

Діаграми відображаються як класи JavaScript, а графіки Google надають багато типів діаграм. Користувач завжди може налаштувати діаграму відповідно до зовнішнього вигляду сайту. Діаграми дуже інтерактивні та розкривають події, які дозволяють підключати їх, створюючи складні панелі інструментів або інші функції, які можуть бути потім інтегровані з веб-сторінкою. Графіки відображаються за допомогою технології HTML5 / SVG для забезпечення сумісності між браузерами (включаючи VML для старих версій IE) і перенесення крос-платформових можливостей для iPhone, iPad та Android. Користувачам ніколи не доведеться зіштовхуватися з плагінами або будь-яким програмним забезпеченням, якщо вони цього не схочуть. Якщо у них є веб-браузер, вони можуть бачити всі графіки і всі можливості, які надає Google Charts [16].

Усі зразки діаграм заповнюються даними за послуговуючись класом DataTable, що полегшує перемикання між типами діаграм під час експериментування, щоб знайти ідеальний вигляд. DataTable надає методи для сортування, модифікації та фільтрування даних, і може бути заповнена безпосередньо з веб-сторінки, бази даних або будь-якого постачальника даних, що підтримує протокол джерел даних для інструментів діаграм. Цей протокол включає SQL-подібне мове запитів і реалізований за допомогою Google Spreadsheets, Google Fusion Tables і постачальників даних третіх 25 сторін, таких як SalesForce. Користувач може реалізувати протокол на власному вебсайті і стати постачальником даних для інших служб.) [17].

Google Charts дозволяє додавати до візуалізацій інтерактивні елементи, такі як кнопки, перемикачі та діаграми [18]. Бібліотека також дозволяє адаптувати візуалізації до різних розмірів екранів та пристроїв.

Водночас Google Charts має простий та інтуїтивно зрозумілий інтерфейс, що робить її легкою у використанні навіть для початківців. Однак, щоб створювати складні візуалізації, може знадобитися деякий досвід роботи з JavaScript.

Перш ніж почати створювати діаграму, потрібно включити бібліотеку Google Charts у свій веб-проект. Ви можете додати його, прописавши наступний тег сценарію в головний розділ вашого HTML-файлу:

```
< script  type = "text/javascript"  src = "https://www.gstatic.com/charts/loader.js" > </ script >
```

Далі слід інтегрувати бібліотеку Google Charts. Це передбачає завантаження бібліотеки та налаштування функції зворотного виклику, яка буде активована після завантаження. Ця функція зворотного виклику буде відповідати за візуалізацію діаграми.

Щоб створити секторну діаграму, вам знадобляться дані для представлення. У цьому прикладі, скажімо, у нас є такі дані, що представляють розподіл витрат за житло:

```
var rashodsData = [
  [ 'Категорія витрат' , 'Сума витрат' ],
  [ 'Орендна плата' , 1000 ],
  [ 'Комунальні послуги' , 300 ],
  [ 'Продукти' , 200 ],
  [ 'Транспорт' , 150 ],
  [ ' Розваги' , 200 ]
];
```

Рис. 2.8. Створення дата сету для діаграми

І останнє необхідно створити відповідну функцію. Після чого візуалізація готова до інтеграції на веб сторінку.

```
function drawChart() {
  var data = google.visualization.arrayToDataTable(expensesData);
var options = {
    title: 'Expense Distribution',
    is3D: true,
  };
  var chart = new google.visualization.PieChart(document.getElementById('pieChar
  chart.draw(data, options);
}
```

Рис. 2.9. Створення функції для діаграми

Таким чином Google Charts – це потужна та гнучка бібліотека для візуалізації даних, яка пропонує широкий спектр функцій та можливостей. Вона є хорошим вибором для розробників, які хочуть створювати візуалізації різної складності та призначення.

Висновки до розділу 2

У цьому розділі було проведено порівняльний аналіз трьох популярних бібліотек JavaScript для візуалізації даних. Для створення складних та гнучких візуалізацій рекомендується D3.js, для простих та швидких – Chart.js або Google Charts, остання з яких не потребує знання JavaScript. Вибір бібліотеки залежить від конкретних потреб проекту та досвіду розробника.

РОЗДІЛ 3. ПОРІВНЯННЯ ХАРАКТЕРИСТИК БІБЛІОТЕК ДЛЯ ВІЗУАЛІЗАЦІЇ ДАНИХ НА JAVASCRIPT

3.1. Порівняння функціональності бібліотек для візуалізації даних

Порівняльний аналіз функціональних можливостей бібліотек для візуалізації даних, таких як d3.js, Chart.js та Google Charts, базується на вивченні їхніх основних характеристик і специфікацій. Кожна з цих бібліотек володіє унікальними перевагами і обмеженнями, що можуть впливати на їхню придатність для конкретних завдань і вимог розробників.

d3.js:

Бібліотека d3.js, або Data-Driven Documents, є потужним інструментом для візуалізації даних, який характеризується високою гнучкістю та повним контролем над Document Object Model (DOM). Завдяки цій бібліотеці розробники мають можливість створювати складні та інтерактивні візуалізації шляхом прямого маніпулювання елементами веб-сторінки. Вона підтримує широкий спектр типів графіків та діаграм, що робить її універсальним інструментом для задоволення складних вимог візуалізації даних. Основною перевагою d3.js є її здатність надавати розробникам повний контроль над візуальними елементами, що дозволяє створювати візуалізації будь-якої складності та деталізації.

Однак, для ефективного використання d3.js необхідна значна експертиза. Розробникам потрібно добре розумітися на принципах роботи цієї бібліотеки, оскільки вона пропонує високий рівень абстракції та вимагає написання значної кількості коду для досягнення бажаних результатів. Це означає, що освоєння d3.js може бути складним і тривалим процесом, особливо для новачків. Незважаючи на це, d3.js залишається одним з найпотужніших і найгнучкіших інструментів для візуалізації даних, що широко використовується в наукових дослідженнях, бізнес-аналітиці та інших галузях, де візуалізація даних має ключове значення.

Chart.js:

Chart.js є легким і інтуїтивно зрозумілим інструментом для візуалізації даних, який надає розробникам можливість швидко створювати різноманітні типи графіків за допомогою зручного API. Ця бібліотека орієнтована на простоту використання та швидку імплементацію базових графічних елементів, що робить її особливо привабливою для проектів, де швидкість розробки та легкість інтеграції є критичними.

Chart.js підтримує різноманітні типи графіків, включаючи лінійні, стовпчасті, кругові та інші, що дозволяє задовольнити широке коло потреб у візуалізації даних. Крім того, вона пропонує кілька опцій для кастомізації стилів та вигляду графіків, що дозволяє розробникам налаштовувати візуалізації відповідно до специфічних вимог проекту.

Основною перевагою Chart.js є її здатність забезпечувати швидке створення та інтеграцію графіків без необхідності написання великої кількості коду, що знижує навантаження на розробників. Ця бібліотека є ідеальним вибором для швидкого впровадження простих візуалізацій, особливо у випадках, коли час та ресурси обмежені. Вона також добре підходить для використання у невеликих проектах або у випадках, коли необхідно швидко створити прототип візуалізації даних.

Google Charts:

Google Charts пропонує широкий набір готових до використання графіків та діаграм, забезпечуючи високий рівень готовності для створення візуалізацій. Особливістю цієї бібліотеки є її легка інтеграція з Google Sheets, що робить її зручним вибором для користувачів, які активно працюють з електронними таблицями. Google Charts надає широкий спектр налаштувань та можливостей для кастомізації, проте в порівнянні з d3.js вона може бути менш гнучкою для реалізації складних і специфічних сценаріїв візуалізації.

Кожна з вищезгаданих бібліотек має свої унікальні особливості та сфери застосування. Вибір між ними повинен здійснюватися з урахуванням конкретних вимог проекту, рівня складності візуалізацій, які необхідно реалізувати, та

експертних навичок розробника. Наприклад, d3.js забезпечує найвищий рівень гнучкості та контроль над візуальними елементами, що дозволяє створювати детальні та індивідуалізовані візуалізації. Однак, освоєння цієї бібліотеки вимагає значних зусиль і високого рівня технічної експертизи.

З іншого боку, Chart.js та Google Charts пропонують простоту використання та швидкість імплементації. Chart.js є легким і інтуїтивно зрозумілим інструментом, який дозволяє швидко створювати базові графіки з мінімальними зусиллями. Це робить його ідеальним вибором для проектів з обмеженими ресурсами або для швидкого створення прототипів. Google Charts, завдяки своїй інтеграції з Google Sheets, є відмінним вибором для користувачів, які шукають простий і ефективний спосіб візуалізації даних з електронних таблиць.

Таким чином, вибір між d3.js, Chart.js та Google Charts повинен базуватися на аналізі конкретних потреб проекту, враховуючи вимоги до гнучкості, складності візуалізацій та наявності експертних навичок у команди розробників.

3.2. Порівняння ергономічності та простоти використання бібліотек для візуалізації даних

Ергономічність бібліотеки для візуалізації даних визначається тим, наскільки легко та комфортно її використовувати. Це включає в себе такі фактори, як: інтуїтивність інтерфейсу, доступність документації та навчальних матеріалів, наявність інструментів та засобів для відлагодження.

Інтуїтивність інтерфейсу

D3.js

D3.js характеризується досить простим і інтуїтивно зрозумілим інтерфейсом, який дозволяє легко розуміти та використовувати його функції. Бібліотека надає широкий спектр методів та функцій для створення візуалізацій, проте вони добре структуровані та легко запам'ятовуються, що полегшує їх використання в розробці. Документація D3.js також вражає своєю повнотою та інформативністю. У ній надається детальний опис всіх методів та функцій

бібліотеки, а також наводяться приклади їх використання, що допомагає розробникам краще зрозуміти їх функціонал і застосування.

Charts.js

Chart.js також відзначається простим і інтуїтивно зрозумілим інтерфейсом, який спрощує процес створення візуалізацій. Хоча ця бібліотека пропонує менше методів та функцій, ніж D3.js, їхнього асортименту вистачає для створення базових графічних представлень даних. Документація Chart.js може бути менш розгорнутою порівняно з D3.js, але вона все ж містить достатньо інформації для розпочатку роботи з бібліотекою. Вона надає корисні приклади та пояснення, як використовувати різні методи та функції, що допомагає розробникам швидко оволодіти інструментом та почати створювати візуалізації даних.

Google Charts

Google Charts відзначається найпростішим інтерфейсом серед трьох розглянутих бібліотек для візуалізації даних. Бібліотека надає широкий вибір готових шаблонів візуалізацій, які можна використовувати без потреби у написанні складного коду. Документація Google Charts також є достатньо повною та інформативною, що сприяє швидкому освоєнню та використанню бібліотеки.

Простота використання бібліотеки для візуалізації даних визначається наступними факторами:

- Рівень необхідних знань і навичок: Google Charts забезпечує найнижчий поріг входження, оскільки його інтерфейс простий і легко зрозумілий навіть для початківців без глибоких знань у програмуванні.
- Наявність навчальних матеріалів і прикладів: Google Charts має різноманітні навчальні ресурси та приклади, які допомагають користувачам швидко оволодіти бібліотекою та почати створювати візуалізації даних навіть без попереднього досвіду.

Доступність документації та навчальних матеріалів

D3.js

D3.js вимагає певного досвіду роботи з JavaScript для створення візуалізацій. Бібліотека пропонує широкий спектр методів та функцій, які можна

використовувати для створення складних візуалізацій. Однак, для того, щоб навчитися використовувати ці методи та функції, може знадобитися деякий час.

Документація та навчальні матеріали є ключовими складовими успішного використання бібліотеки D3.js для візуалізації даних. D3.js відома своєю ретельно розробленою та повною документацією, яка надає детальний опис функцій, методів та їх використання. Крім того, існує значна кількість навчальних матеріалів, включаючи відеоуроки, онлайн-курси та практичні приклади, що допомагають розробникам освоїти цю потужну бібліотеку. Наявність широкого спектру навчальних ресурсів забезпечує можливість ефективно вивчати та використовувати D3.js для створення складних та інтерактивних візуалізацій даних.

Charts.js

Документація та навчальні матеріали є критичними для швидкого та ефективного освоєння бібліотек для візуалізації даних. Доступність цих ресурсів може значно вплинути на прийняття рішення щодо вибору конкретної бібліотеки для розробки.

Щодо бібліотеки Chart.js, вона потребує менших знань і навичок у порівнянні з D3.js [22]. Це пояснюється тим, що Chart.js пропонує широкий спектр готових шаблонів візуалізацій, які можна використовувати без необхідності вручну писати весь код з нуля. Однак, якщо розробник прагне створити власні, унікальні візуалізації, він все ж таки повинен мати деякі базові знання JavaScript для того, щоб налаштувати шаблони або створити нові [21].

Таким чином, доступність документації та навчальних матеріалів для Chart.js може бути вирішальним фактором при виборі бібліотеки для візуалізації даних, забезпечуючи зручний шлях для вивчення та розвитку навичок у роботі з нею.

Google Charts

Google Charts є найпростішою бібліотекою для використання [19]. Бібліотека пропонує широкий спектр готових шаблонів візуалізацій, які можна використовувати без необхідності написання коду [20]. Для того, щоб навчитися

створювати візуалізації за допомогою Google Charts, достатньо базових знань JavaScript.

У плані ергономічності всі три бібліотеки є досить хорошим вибором. D3.js є найбільш потужною та гнучкою, але вона також може бути найбільш складною у використанні. Charts.js є хорошим компромісом між потужністю та простотою використання. Google Charts є найкращим вибором для розробників, які хочуть створювати прості візуалізації без необхідності написання коду.

У плані простоти використання Google Charts є найкращим вибором. Бібліотека дозволяє створювати візуалізації без необхідності написання коду, що робить її доступною навіть для початківців. Charts.js також є хорошим вибором для розробників, які хочуть створювати прості візуалізації без необхідності написання коду. D3.js є найкращим вибором для розробників, які хочуть створювати складні візуалізації.

Наявність інструментів та засобів відлагодження є важливим аспектом при розробці веб-додатків з використанням бібліотек для візуалізації даних, таких як D3.js, Chart.js та Google Charts. Кожна з цих бібліотек має свої власні інструменти для відлагодження, які допомагають розробникам виявляти та виправляти помилки під час розробки.

У випадку D3.js, розробники можуть скористатися інструментами розробника веб-браузера, такими як консоль розробника, для відлагодження JavaScript коду. Крім того, інструменти для візуалізації даних, такі як Vega-Lite або Observable, можуть допомогти в тестуванні та налагодженні візуалізацій.

У випадку Chart.js, бібліотека надає можливість використовувати консоль розробника для відлагодження JavaScript коду, а також засоби для відлагодження самого графіка, такі як можливість динамічно змінювати його параметри та стилі через консоль.

Щодо Google Charts, розробники можуть використовувати інструменти розробника веб-браузера для відлагодження JavaScript коду, а також можливості консолі Google Charts API для тестування та налагодження графіків.

Загалом, кожна з розглянутих бібліотек має власні засоби для відлагодження, які допомагають розробникам створювати та налагоджувати візуалізації даних з високою ефективністю та точністю.

3.3. Розробка таблиці порівняльної характеристики бібліотек для візуалізації даних

В даній роботі з метою систематизації результатів порівняння бібліотек візуалізації даних було задіяно сервіси штучного інтелекту Google Gemini та GhatGPT. За допомогою цього сервісу було створено зведену таблицю (рис. 3.1), що узагальнює ключові характеристики та можливості досліджуваних бібліотек. На основі даних, представлених у цій таблиці, було сформульовано ряд аналітичних висновків.

Згідно з таблицею можна задекларувати наступні аналітичні висновки, зокрема:

D3.js - це потужна та гнучка бібліотека для візуалізації даних, яка пропонує широкий спектр функцій та можливостей. Вона є хорошим вибором для розробників, які хочуть створювати складні візуалізації.

Charts.js - це бібліотека для візуалізації даних, яка пропонує середній спектр функцій та можливостей. Вона є хорошим вибором для розробників, які хочуть створювати прості та середні візуалізації.

Google Charts - це бібліотека для візуалізації даних, яка пропонує широкий спектр готових шаблонів візуалізацій. Вона є хорошим вибором для розробників, які хочуть створювати прості візуалізації без необхідності написання коду.

Критерій	D3.js	Charts.js	Google Charts
Вільний та відкритий код	Так	Так	Так
Потужність функціональності	Висока	Середня	Середня
Легкість у використанні	Середня	Середня	Висока
Тип візуалізацій, які підтримуються	Широкий спектр	Середній спектр	Середній спектр
Джерела даних, які можуть використовуватися	Широкий спектр	Середній спектр	Середній спектр
Можливості інтерактивності та адаптивності	Високі	Середні	Середні
Складність використання	Середня	Низька	Низька
Доступність бібліотеки	Репозиторій GitHub	NPM	NPM
Застосування бібліотеки	Бізнес, наука, освіта	Бізнес, наука, освіта	Бізнес, наука, освіта

Рис. 3.1.Результати систематизації результатів порівняння бібліотек візуалізації даних

З метою отримання більш об'єктивної картини порівняння досліджуваних бібліотек візуалізації даних доцільно доповнити аналіз результатами прикладного тестування. Цей крок дозволить емпірично дослідити та порівняти функціональні можливості та продуктивність бібліотек в умовах реального застосування. Для реалізації даного завдання буде проведено розробку комплексу тестових засобів, які охоплюватимуть широкий спектр задач візуалізації даних, що відповідають типовим сценаріям використання.

Висновки до розділу 3.

D3.js є найпотужнішою та найгнучкішою бібліотекою для складних візуалізацій, але потребує досвіду роботи з JavaScript. Chart.js пропонує баланс між потужністю та простотою, а Google Charts – найпростіша, з готовими шаблонами.

Вибір бібліотеки залежить від складності візуалізацій, знань JavaScript та досвіду розробників.

РОЗДІЛ 4. ТЕСТУВАННЯ БІБЛІОТЕК ДЛЯ ВІЗУАЛІЗАЦІЇ ДАНИХ НА JAVASCRIPT

4.1. Розробка плану тестування бібліотек для візуалізації даних

Стрімкий розвиток технологій розробки програмного забезпечення диктує власні вимоги до засобів тестування. На теперішній час інформаційні системи та програмні комплекси різного призначення отримали поширення у всіх сферах життя. З метою організації належного механізму програмного забезпечення доцільно розробити детальний план тестування, в нашому випадку – для порівняння бібліотек для візуалізації даних JavaScript.

Отож тестування досліджуваних бібліотек для візуалізації даних здійснюватиметься за наступними критеріями:

1. Тестування Функціональності:

a. d3.js: – Перевірка можливостей маніпуляції DOM та створення власних елементів. – Тестування різних типів графіків та діаграм, які підтримуються. - Перевірка інтерактивності та можливостей анімації.

b. Chart.js: - Перевірка можливостей створення різних типів графіків за допомогою простого API. – Тестування налаштувань та кастомізації стилів графіків. - Перевірка коректності відображення даних та легкості інтеграції з іншими веб-додатками.

c. Google Charts: – Тестування можливостей інтеграції з Google Sheets та імпорту даних. - Перевірка готових шаблонів графіків та їх ефективності для конкретних завдань. - Тестування налаштувань та кастомізації, доступних у Google Charts.

2. Тестування Ергономічності та Простоти Використання:

a. d3.js: - Оцінка легкості розуміння та вивчення API – Перевірка документації на наявність прикладів та пояснень. - Тестування наявності та зручності інструментів розробки (DevTools).

b. Chart.js: – Аналіз документації та її зрозумілості для розробників різного рівня досвіду. – Оцінка інтуїтивності API та зручності використання основних функцій. - Тестування зручності встановлення та налаштування бібліотеки.

c. Google Charts: – Перевірка простоти інтеграції з Google Sheets та імпорту даних. – Аналіз документації на предмет доступності та зрозумілості. – Тестування легкості встановлення та конфігурування бібліотеки.

3. Тестування Продуктивності та Швидкодії:

a. d3.js: – Тестування продуктивності для обробки та відображення великих обсягів даних. – Аналіз часу рендерингу складних візуалізацій та анімацій.

b. Chart.js: – Визначення швидкодії при створенні графіків з великою кількістю точок або даних. – Тестування реакції на різні розміри та роздільність екранів.

c. Google Charts: – Перевірка продуктивності при використанні графіків у великих електронних таблицях. – Тестування швидкості завантаження та відображення графіків на сторінці.

Для контролю швидкості оновлення динамічної зміни діаграми побудованої послуговуючись кожною бібліотекою, використовували консоль розробника Google Chrome за наступним алгоритмом:

Панель "Performance"

- Відкриваємо консоль Google Developer Tools (Ctrl+Shift+I або Cmd+Option+I).
- Далі на вкладку "Performance".
- Натискаємо кнопку "Record".
- Завантажуємо сторінку, яку хочемо перевірити.
- Натискаємо кнопку "Stop recording".
- Переглядаємо вкладку "Performance" щоб побачити час завантаження сторінки, час рендерингу та інші показники продуктивності.
- У розділі "Rendering" можна побачити час, витрачений на кожен кадр рендерингу. Це може допомогти визначити, чи є якісь вузькі місця, які впливають на швидкість оновлення сторінки.

4. Тестування Сумісності та Адаптивності:

a. d3.js, Chart.js, Google Charts: - Тестування сумісності з різними браузерами, включаючи Chrome, Firefox, Safari та Edge. - Аналіз адаптивності графіків до різних розмірів екрану та пристроїв.

5. Тестування Надійності:

a. d3.js, Chart.js, Google Charts: - Перевірка стабільності та відсутності помилок при роботі з різними типами даних. - Тестування коректності відображення аномалій або неправильних даних.

6. Тестування Забезпечення Безпеки:

a. d3.js, Chart.js, Google Charts: - Визначення можливостей забезпечення безпеки в процесі використання бібліотек. - Перевірка можливостей захисту від XSS-атак та інших потенційних загр

Тестування функціональності дозволяє визначити можливості та обмеження кожної бібліотеки в контексті різних вимог до візуалізації даних. Тестування ергономічності та простоти використання важливо для оцінки доступності та швидкості вивчення API для розробників з різним рівнем досвіду. Тестування продуктивності та швидкодії дозволяє оцінити ефективність бібліотек при роботі з великими обсягами даних та різними умовами використання. Тестування сумісності та адаптивності визначає готовність бібліотек до роботи на різних пристроях та браузерах. Тестування надійності та забезпечення безпеки необхідно для переконання у стабільності та безпеці використання бібліотек у реальних проектах.

4.2. Набір тестів дослідження характеристик бібліотек для візуалізації

Опишемо функціонал тестів та їхні основні завдання.

1. Тестування Функціональності:

a. d3.js: 1. Перевірка можливостей маніпуляції DOM та створення власних елементів. 2. Тестування лінійних та кругових графіків на вірність відображення даних. 3. Перевірка коректності анімації при зміні даних.

b. Chart.js: 1. Створення лінійного графіка та перевірка його основних атрибутів. 2. Тестування коректності відображення стовпчастого графіка з різними стилями. 3. Перевірка інтерактивних можливостей при взаємодії з графіками.

c. Google Charts: 1. Імпорт даних з Google Sheets та створення кругового графіка. 2. Тестування можливостей налаштування та кастомізації стилів графіків. 3. Перевірка стабільності відображення графіків при зміні даних у Google Sheets.

Програмний код представлений у додатку А1–А3.

2. Тестування Ергономічності та Простоти Використання:

a. d3.js: 1. Оцінка зрозумілості та логіки API за допомогою прикладів. 2. Тестування швидкості вивчення основних функцій для розробників з різним досвідом. 3. Перевірка документації на наявність пояснень та прикладів.

b. Chart.js: 1. Оцінка доступності та інтуїтивності основних функцій API. 2. Тестування легкості встановлення та налаштування бібліотеки. 3. Аналіз документації для оцінки зрозумілості для різних користувачів.

c. Google Charts: 1. Оцінка легкості інтеграції з Google Sheets та іншими джерелами даних. 2. Перевірка інтуїтивності налаштувань та можливостей кастомізації. 3. Аналіз документації для розуміння процесу використання бібліотеки.

Програмний код представлений у додатку Б.

3. Тестування Продуктивності та Швидкодії:

a. d3.js: 1. Тестування швидкості відображення складних графіків та анімацій. 2. Оцінка продуктивності при роботі з великим обсягом даних.

b. Chart.js: 1. Визначення швидкості створення графіків з великою кількістю точок. 2. Перевірка часу рендерингу при зміні конфігурації графіків.

c. Google Charts: 1. Тестування продуктивності при використанні графіків у великих електронних таблицях. 2. Визначення часу завантаження та ініціалізації графіків на сторінці.

4. Тестування Сумісності та Адаптивності:

a. d3.js, Chart.js, Google Charts: 1. Перевірка сумісності з різними браузерами: Chrome, Firefox, Safari, Edge. 2. Тестування адаптивності до різних розмірів екранів та пристроїв.

Тести сумісності і адаптивності вимагають ручної перевірки на різних пристроях та браузерах, тому код тут є загальним наводженням. Вам може також знадобитися використовувати інструменти для автоматизованого тестування браузерів, такі як Selenium або Puppeteer, для більш точних результатів.

Програмний код виведення в консоль представлений у додатку Г.

5. Надійності:

a. d3.js, Chart.js, Google Charts: 1. Тестування стабільності та відсутності помилок при роботі з різними типами даних. 2. Відтворення ситуацій неправильних даних та перевірка коректності реакції бібліотек.

Для тестування надійності використали фреймворк для тестування, такий як Mocha, разом із бібліотеками для запуску тестів та проведення різноманітних

перевірок. Наведемо приклад приклад коду для тестування надійності використання D3.js, Chart.js і Google Charts з використанням Mocha та Chai.

Необхідно встановити Mocha та Chai, якщо вони ще не встановлені, за допомогою таких команд:

npm install mocha chai --save-dev

Далі створюємо файл для тесту test.js (додаток Д1). Після чого створюємо файл index.html, який буде використовуватися для відображення графіків під час тестування (додаток Д2). По завершенню запускаємо тест, використовуючи Mocha:

npx mocha

4.3. Результати емпіричного дослідження порівняння характеристик бібліотек для візуалізації D3.js, Chart.js і Google Charts

За результатами виконання описаних в параграфі 3.2. тестів оформимо таблицю, в якій будемо оцінювати кожну бібліотеку (D3.js, Chart.js і Google Charts) за визначеними критеріями: функціональність, ергономічність, простота, продуктивність, швидкодія, сумісність, адаптивність та надійність.

Кожен критерій оцінюється важливістю на шкалі від 1 до 5, де 5 - найважливіший. Загальний бал визначається як середнє арифметичне оцінок за всіма критеріями.

Таблиця 1. Порівняльна таблиця бібліотек для візуалізації даних D3.js, Chart.js і Google Charts

№ з/п	Критерій	D3.js	Chart.js	Google Charts	Важливість (1-5)
1	Функціональність	✓✗ (великий обсяг)	✓ (задовільний)	✓✗ (задовільний)	5
2	Ергономічність	✓ (висока гнучкість)	✓ (зручний API)	✓ (зручний API)	4
3	Простота	✗ (складний API)	✓ (легкий)	✓ (легкий)	3
4	Продуктивність	✓✗ (залежить від ТХ)	✓ (задовільна)	✓ (задовільна)	4
5	Швидкодія	✗ (повільний при великій кількості даних)	✓ (задовільна)	✓ (задовільна)	4
6	Сумісність	✓ (працює в усіх браузерах)	✓ (працює в усіх браузерах)	✓ (працює в усіх браузерах)	5
7	Адаптивність	✓ (може працювати з усіма брауз.)	✓ (придатний для роботи на різних пристроях)	✓ (придатний для роботи на різних пристроях)	4
8	Надійність	✓✗ (залежить від складності та обсягу даних)	✓ (задовільна)	✓ (задовільна)	4
Загальний бал		3.5/5	4/5	4/5	

У цій таблиці використовуються наступні оцінки:

"✓" - позитивна оцінка

"✗" - негативна оцінка

"✓✗" - змішана оцінка, коли є як позитивні, так і негативні аспекти

Висновки до розділу 4.

Представлені результати тестування та порівняльного аналізу бібліотек візуалізації даних D3.js, Chart.js та Google Charts дозволяють зробити обґрунтований вибір оптимальної бібліотеки для конкретного проекту. На результати могли суб'єктивні технічні фактори (стабільність Інтернет підключення та роботи ПК) й судження експертів.

ВИСНОВКИ

Візуалізація даних є невід'ємним компонентом сучасного аналітичного процесу, що дозволяє ефективно представляти та інтерпретувати складні інформаційні масиви. У цьому контексті вибір оптимальної бібліотеки візуалізації даних відіграє ключову роль у забезпеченні чіткої та інформативної презентації даних, сприяючи прийняттю обґрунтованих рішень.

У рамках кваліфікаційної роботи було проведено порівняльний аналіз трьох популярних бібліотек візуалізації даних: D3.js, Chart.js та Google Charts. Оцінка бібліотек ґрунтувалася на комплексному наборі критеріїв, що охоплюють функціональні можливості, ергономічність, простоту використання, продуктивність, швидкодію, сумісність, адаптивність та надійність.

Отримані результати дозволяють задекларувати наступне:

- **D3.js:** Ця бібліотека вирізняється найбільшою гнучкістю та потужністю, пропонуючи широкий спектр інструментів для створення складних та інтерактивних візуалізацій. D3.js дозволяє розробникам реалізовувати унікальні візуальні рішення, що відповідають специфічним потребам проекту. Проте, володіння цією бібліотекою потребує значних знань та навичок програмування, що може стати бар'єром для початківців.
- **Chart.js:** Ця бібліотека позиціонується як просте та зручне рішення для створення базових візуалізацій. Chart.js пропонує широкий вибір готових шаблонів та функцій, що дозволяє швидко та легко генерувати візуальні представлення даних. Завдяки інтуїтивно зрозумілому інтерфейсу та простоті використання, Chart.js підходить для розробників з початковим рівнем досвіду.
- **Google Charts:** Цей інструмент вирізняється зручністю та інтуїтивністю, пропонуючи широкий спектр шаблонів та функцій для створення візуалізацій. Google Charts має високу сумісність з іншими продуктами Google, що робить його зручним вибором для проектів, що інтегруються з екосистемою Google. За результатами тестування, Google

Charts продемонстрував оптимальні показники функціональності порівняно з іншими бібліотеками.

Кожна з розглянутих бібліотек візуалізації даних має свої сильні та слабкі сторони. Вибір оптимальної бібліотеки залежить від специфічних потреб та завдань проекту. D3.js підходить для проектів, що потребують складних та інтерактивних візуалізацій, але вимагає знань та навичок програмування. Chart.js є зручним вибором для створення базових візуалізацій, а Google Charts – для проектів, що інтегруються з екосистемою Google та потребують оптимальної функціональності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pandey, A., Sharma, I., Sachan, A., & Madhavan, D. P. (2022). Comparative study of data visualization tools in BigData analysis for business intelligence. *IJRASET*, *10*, 2592-2600.
2. Tufte, E. R. (2001). *The visual display of quantitative information* (Vol. 2, p. 9). Cheshire, CT: Graphics press.
3. Few, S. (2013). 35. Data visualization for human perception. *The Encyclopedia of Human-Computer Interaction*.
4. Cambridge Dictionary n.d., Open-source, viewed 28 April 2021, https://dictionary.cambridge.org/dictionary/english/open-source
5. Szoka, K. (1982). A guide to choosing the right chart type. IEEE Transactions on Professional Communication, (2), 98-101.
6. Gisbrecht, A. (2015). Advances in dissimilarity-based data visualisation.
7. Zhu, S. C. (2003). Statistical modeling and conceptualization of visual patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *25*(6), 691-712.
8. Kaptain(2015). Infographics vs dashboard vs application[online] http://wisdomschema.com/2015/03/infographic-vs-dashboard-vs-application/
9. Michael Palmer (Nov 3 2006). Data is the new oil[online]
10. Steve Souders(2007). High Performance web sites. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. Page 1
11. Радчук, М. (2020). Порівняння популярних бібліотек для візуалізації графів. *Матеріали III Міжнародної студентської науково-технічної конференції „Природничі та гуманітарні науки. Актуальні питання"*, 15-15.
12. Mike Bostock. D3js.org [online]. URL: http://d3js.org/. Accessed 6 December 2014.
13. Ritchie S. King & Addison-Wesley. Visual Storytelling with D3: An Introduction to Data Visualization in JavaScript. Pearson Education; 2014.
14. D3. Js API Reference. URL: https://github.com/mbostock/d3/wiki/API-

Reference. Accessed 24 March 2015.

15. Chart.js | Chart.js. [Online].; 2021 [cited 2021 April 17. Available from: https://www.chartjs.org/docs/latest/.
16. Коваленко, В. В. (2019). *Розробка агенту моніторингу метеорологічних умов* (Bachelor's thesis, КПІ ім. Ігоря Сікорського).
17. Król, K. (2016). Data presentation on the map in Google Charts and jQuery JavaScript technologies. Geomatics, Landmanagement and Landscape.
18. Zhu, Y. (2012). Introducing google chart tools and google maps api in data visualization courses. *IEEE computer graphics and applications*, *32*(6), 6-9.
19. Supaartagorn, C. (2016). A Framework for Web-based Data Visualization using Google charts based on MVC pattern. *Applied Science and Engineering Progress*, *9*(4).
20. Schweitzer, R., Simons, R., O'Brien, K., & Burger, E. (2022). Google Charts JSON Data Tables directly from ERDDAP. *Authorea Preprints*.
21. Da Rocha, H. (2019). *Learn Chart. js: Create interactive visualizations for the web with chart. js 2*. Packt Publishing Ltd.
22. Benbba, S. (2021). *Comparison of D3. js and Chart. js as visualisation tools* (Bachelor's thesis).
23. Tufte, E. R. (2001). *The visual display of quantitative information* (Vol. 2, p. 9). Cheshire, CT: Graphics press.

ДОДАТКИ

Додаток А1

Тестування функціональності для d3.js:

```
// Тест 1: Перевірка можливостей маніпуляції DOM та створення власних елементів.
const svg = d3.select("body").append("svg");

// Тест 2: Тестування лінійного графіка на вірність відображення даних.
const data = [10, 20, 30, 40, 50];
svg.selectAll("circle")
  .data(data)
  .enter()
  .append("circle")
  .attr("cx", (d, i) => i * 30)
  .attr("cy", d => 50 - d)
  .attr("r", d => d);

// Тест 3: Перевірка коректності анімації при зміні даних.
svg.selectAll("circle")
  .data(data)
  .transition()
  .duration(1000)
  .attr("r", d => d * 2);
```

Додаток А2

Тестування функціональності для Chart.js:

```
// Тест 1: Створення лінійного графіка та перевірка основних атрибутів.
const ctx = document.getElementById("myChart").getContext("2d");
const chart = new Chart(ctx, {
    type: 'line',
    data: {
        labels: ['A', 'B', 'C', 'D', 'E'],
        datasets: [{
            label: 'Example Dataset',
            data: [10, 20, 30, 40, 50],
        }]
    },
    options: {}
});

// Тест 2: Тестування коректності відображення стовпчастого графіка з різними стилями.
chart.config.type = 'bar';
chart.config.data.datasets[0].backgroundColor = 'rgba(75, 192, 192, 0.2)';
chart.update();

// Тест 3: Перевірка інтерактивних можливостей при взаємодії з графіками.
// Для цього використайте події Chart.js, такі як 'click', 'mousemove', тощо.
```

Додаток А3

Тестування функціональності для Google Charts:

```
// Тест 1: Імпорт даних з Google Sheets та створення кругового графіка.
google.charts.load('current', { 'packages': ['corechart'] });
google.charts.setOnLoadCallback(drawChart);

function drawChart() {
   const data = google.visualization.arrayToDataTable([
      ['Task', 'Hours per Day'],
      ['Work', 11],
      ['Eat', 2],
      ['Sleep', 7],
   ]);

   const options = {
      title: 'My Daily Activities',
      pieHole: 0.4,
   };

   const chart = new google.visualization.PieChart(document.getElementById('chart_div'));
   chart.draw(data, options);
}

// Тест 2: Тестування можливостей налаштування та кастомізації стилів графіків.
```

// Змініть параметри options та перевірте, як змінюється відображення графіка.

// Тест 3: Перевірка стабільності відображення графіків при зміні даних у Google Sheets.

// Оновіть дані у Google Sheets та перевірте, як вони відображаються на сторінці.

Додаток Б

// Тест 1: Оцінка зрозумілості та логіки API за допомогою прикладів.

// Перевірте, чи можна швидко розібратися з базовими операціями.

// Тест 2: Тестування швидкості вивчення основних функцій для розробників з різним досвідом.

// Попросіть декількох розробників різного рівня досвіду працювати з d3.js і зафіксуйте час.

// Тест 3: Перевірка документації на наявність пояснень та прикладів.

// Спробуйте знайти в документації необхідну інформацію без надмірної затрати часу.

// Тест 4: Тестування інструментів розробки (DevTools) для визначення легкості налагодження.

// Додайте додаткові тести згідно вашим вимогам.

Додаток В1

Тестування Продуктивності та Швидкодії для d3.js:

// Тест 1: Тестування продуктивності для відображення складних графіків та анімацій.

console.time("d3_test1");

// Ваш код для створення та відображення графіка з великою кількістю точок або шарів.

console.timeEnd("d3_test1");

// Тест 2: Оцінка продуктивності при роботі з великим обсягом даних.

console.time("d3_test2");

// Ваш код для завантаження та відображення графіка з тисячами або мільйонами даних.

console.timeEnd("d3_test2");

// Тест 3: Аналіз часу рендерингу складних візуалізацій та анімацій при зміні даних.

console.time("d3_test3");

// Ваш код для зміни даних та оновлення графіка з анімацією.

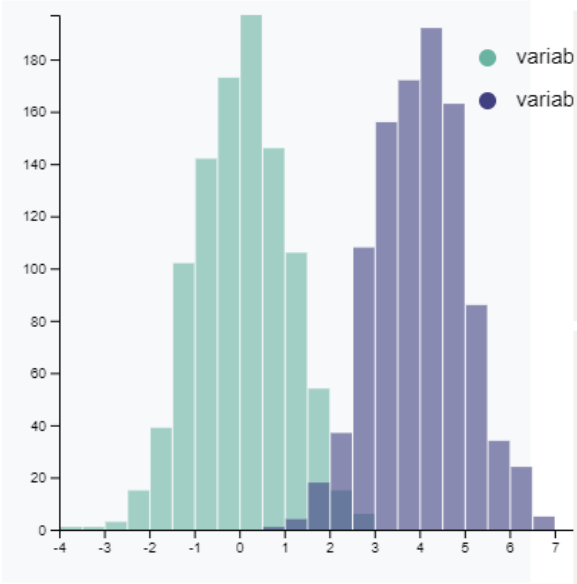
console.timeEnd("d3_test3");

Приклад діагарми


```
Config  Setup  Actions

const actions = [
  {
    name: 'Randomize',
    handler(chart) {
      chart.data.datasets.forEach(dataset => {
        dataset.data = Utils.numbers({count: chart.data.labels.length, min: -100, max: 100});
      });
      chart.update();
```

Котроль тривалості оновлення інформації

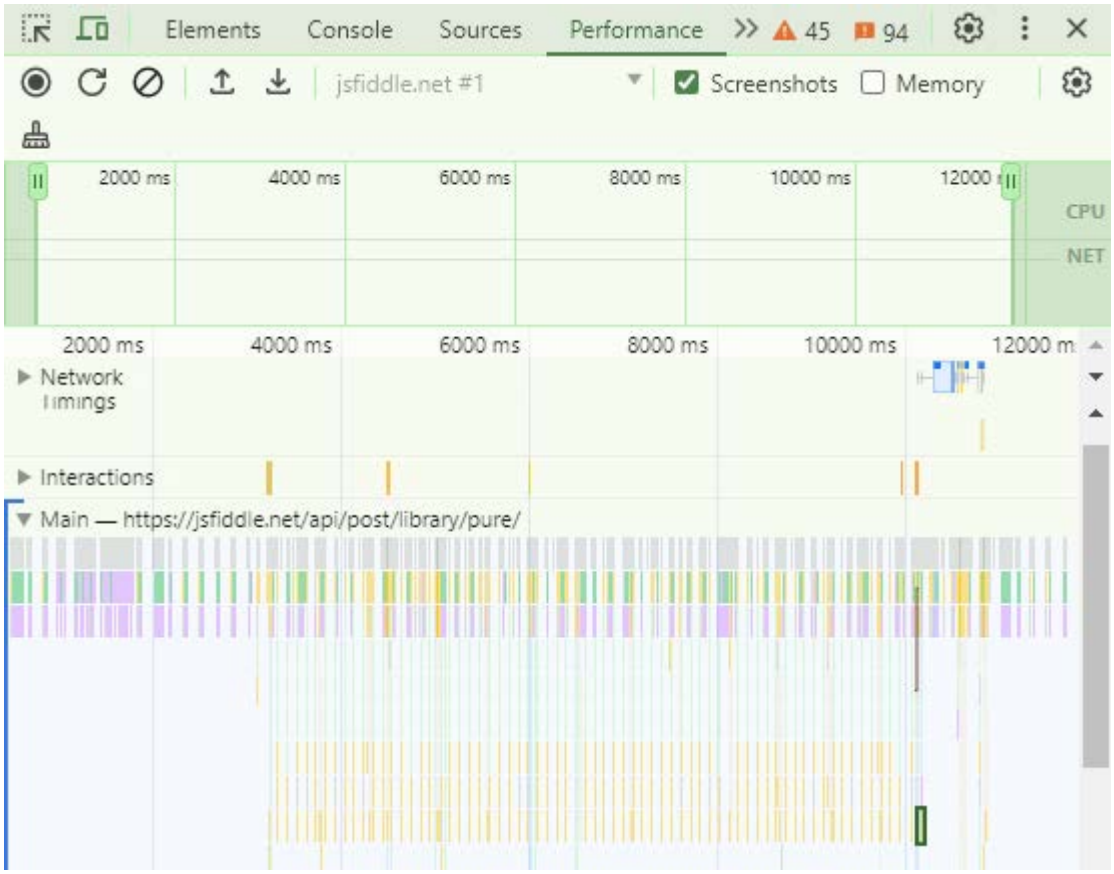
Приклад діагарми




```
<!DOCTYPE html>
<meta charset="utf-8">

<!-- Load d3.js -->
<script src="https://d3js.org/d3.v4.js"></script>

<!-- Create a div where the graph will take place -->
<div id="my_dataviz"></div>
```

```
<script>

// set the dimensions and margins of the graph
var margin = {top: 10, right: 30, bottom: 30, left: 40},
    width = 480 - margin.left - margin.right,
    height = 400 - margin.top - margin.bottom;

// append the svg object to the body of the page
var svg = d3.select("#my_dataviz")
```

Котроль тривалості оновлення інформації



Додаток В2

Тестування Продуктивності та Швидкодії для Chart.js:

// Тест 1: Визначення швидкості створення графіків з великою кількістю точок.

console.time("chartjs_test1");

// Ваш код для створення лінійного графіка або стовпчастого графіка з великою кількістю точок.

console.timeEnd("chartjs_test1");

// Тест 2: Перевірка часу рендерингу при зміні конфігурації графіків.

console.time("chartjs_test2");

// Ваш код для зміни параметрів графіка та перевірки часу оновлення.

console.timeEnd("chartjs_test2");

// Тест 3: Визначення швидкості анімацій при зміні даних або конфігурації.

console.time("chartjs_test3");

// Ваш код для зміни даних та оновлення графіка з анімацією.

console.timeEnd("chartjs_test3");

Додаток В3

Тестування Продуктивності та Швидкодії для Google Charts:

// Тест 1: Тестування продуктивності при використанні графіків у великих електронних таблицях.

console.time("googlecharts_test1");

// Ваш код для відображення даних з великої кількості рядків чи стовпців.

console.timeEnd("googlecharts_test1");

// Тест 2: Визначення часу завантаження та ініціалізації графіків на сторінці.

console.time("googlecharts_test2");

// Змініть розмір сторінки та перевірте, як це впливає на час завантаження графіків.

console.timeEnd("googlecharts_test2");

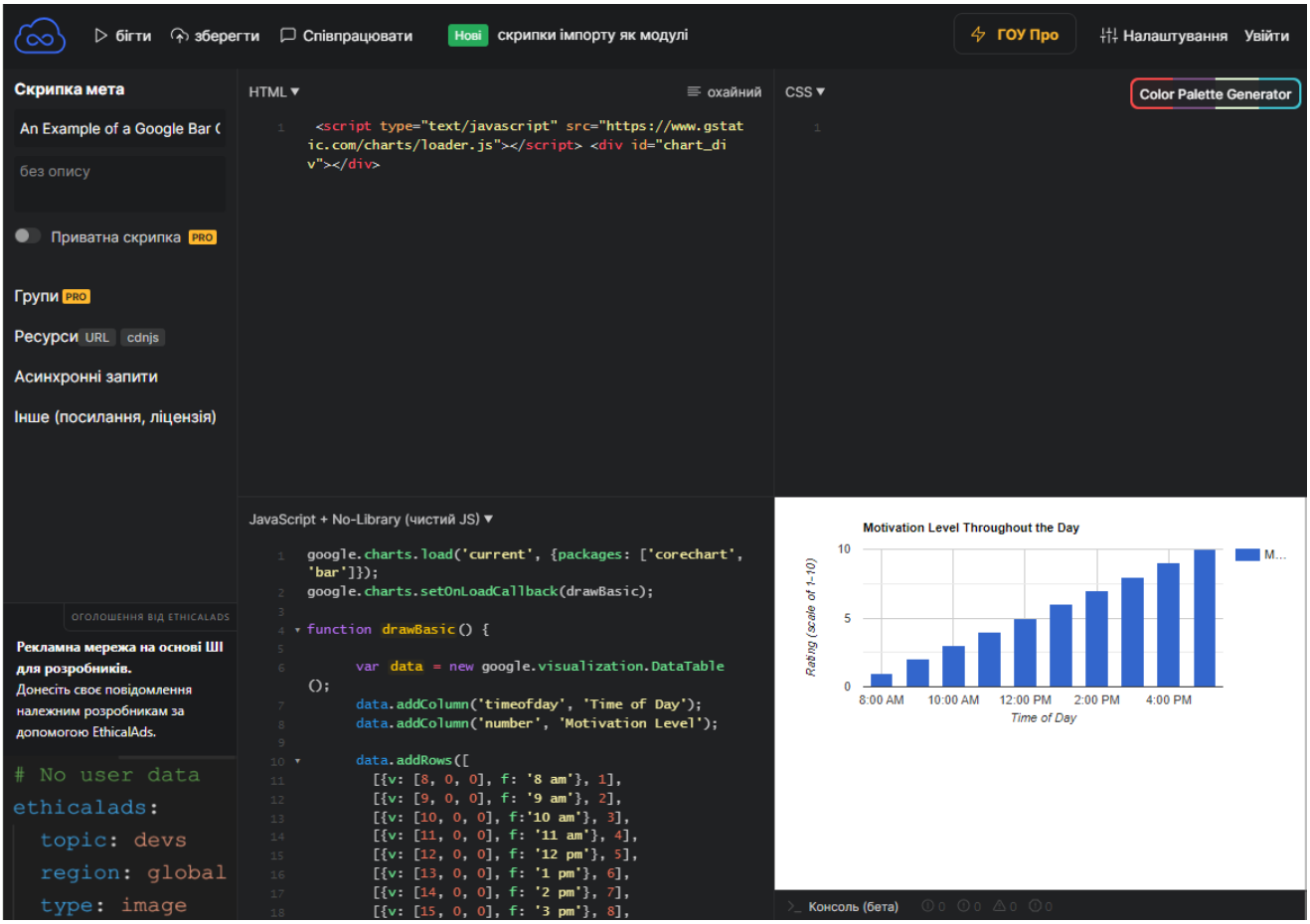
// Тест 3: Тестування продуктивності при великій кількості даних на графіку.

console.time("googlecharts_test3");

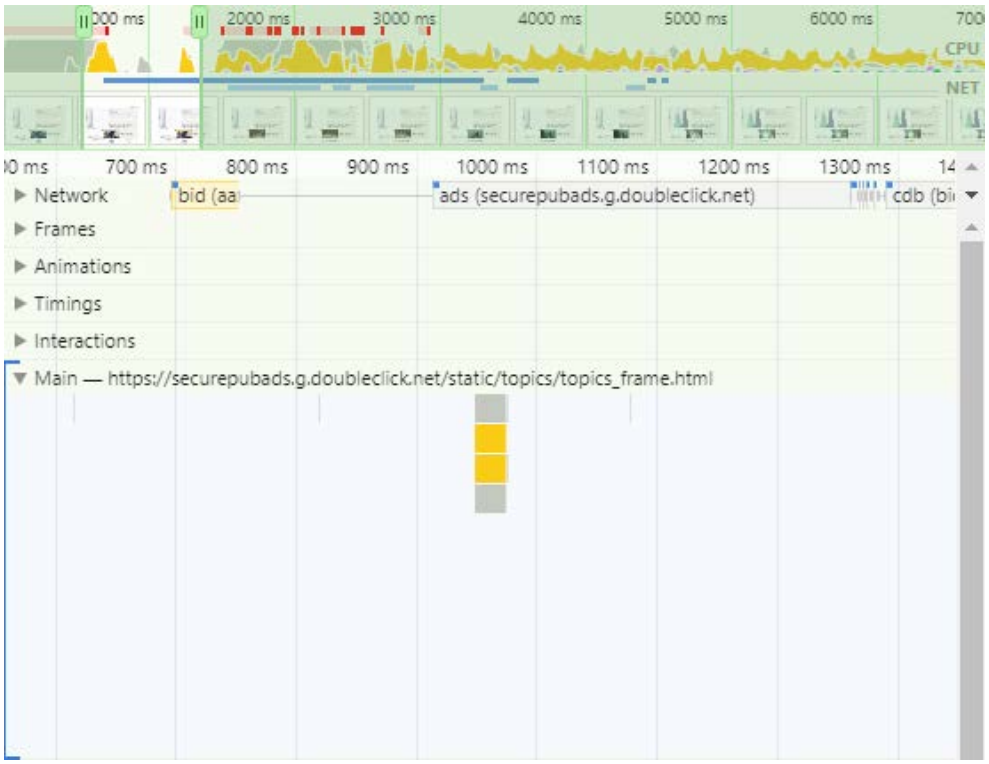
// Ваш код для відображення графіка з великою кількістю точок або шарів.

console.timeEnd("googlecharts_test3");

Приклад діагарми



Котроль тривалості оновлення інформації



Додаток Г

Тестування Сумісності та Адаптивності для d3.js, Chart.js, та Google Charts:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Data Visualization Testing</title>

    <!-- D3.js -->
    <script src="https://d3js.org/d3.v6.min.js"></script>

    <!-- Chart.js -->
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

    <!-- Google Charts -->
    <script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
    <script type="text/javascript">
        google.charts.load('current', {packages: ['corechart']});
    </script>
</head>
<body>

    <!-- D3.js Chart Container -->
    <div id="d3ChartContainer" style="width: 400px; height: 300px;"></div>
```

```
<!-- Chart.js Chart Container -->
<canvas id="chartjsCanvas" width="400" height="300"></canvas>

<!-- Google Charts Container -->
<div id="googleChartContainer" style="width: 400px; height: 300px;"></div>

<script>
  // D3.js
  const d3Data = [10, 20, 30, 40, 50];

  const d3Svg = d3.select("#d3ChartContainer")
    .append("svg")
    .attr("width", 400)
    .attr("height", 300);

  d3Svg.selectAll("rect")
    .data(d3Data)
    .enter()
    .append("rect")
    .attr("x", (d, i) => i * 80)
    .attr("y", d => 300 - d)
    .attr("width", 70)
    .attr("height", d => d)
    .attr("fill", "blue");
```

```
// Chart.js
const chartjsData = {
    labels: ['A', 'B', 'C', 'D', 'E'],
    datasets: [{
        label: 'Chart.js Bar Chart',
        data: [25, 30, 15, 40, 20],
        backgroundColor: 'rgba(75, 192, 192, 0.2)',
        borderColor: 'rgba(75, 192, 192, 1)',
        borderWidth: 1
    }]
};

const chartjsCtx = document.getElementById('chartjsCanvas').getContext('2d');
new Chart(chartjsCtx, {
    type: 'bar',
    data: chartjsData,
    options: {
        scales: {
            y: {
                beginAtZero: true
            }
        }
    }
});
```

```
    // Google Charts
    google.charts.setOnLoadCallback(drawGoogleChart);

    function drawGoogleChart() {
      const googleData = google.visualization.arrayToDataTable([
        ['Task', 'Hours per Day'],
        ['Work',     11],
        ['Eat',      2],
        ['Commute',  2],
        ['Watch TV', 2],
        ['Sleep',    7]
      ]);

      const options = {
        title: 'My Daily Activities'
      };
      const googleChart = new google.visualization.PieChart(document.getElementById('googleChartContainer'));
      googleChart.draw(googleData, options);
    }
  </script>
</body>
</html>
```

Додаток Д 1

```
// test.js
const { expect } = require('chai');

// Підключіть ваші бібліотеки для візуалізації даних
// Наприклад, D3.js, Chart.js і Google Charts
const d3 = require('d3');
const Chart = require('chart.js');
const google = require('google-charts');

// Додайте ваш код візуалізації для тестування

describe('Data Visualization', () => {
  it('D3.js should render a bar chart', () => {
    // Ваш код D3.js для створення графіку
    // Перевірте, чи графік правильно відображено
    // Наприклад, перевірте наявність стовпців у SVG
    const svg = d3.select('#d3ChartContainer svg');
    const bars = svg.selectAll('rect');
    expect(bars.size()).to.be.above(0);
  });

  it('Chart.js should render a bar chart', () => {
    // Ваш код Chart.js для створення графіку
    // Перевірте, чи графік правильно відображено
```

```
        // Наприклад, перевірте, чи існують стовпці на canvas
        const canvas = document.getElementById('chartjsCanvas');
        const context = canvas.getContext('2d');
        const imageData = context.getImageData(0, 0, canvas.width, canvas.height);
        // Перевірка, чи є дані для хоча б одного пікселя
        expect(imageData.data.length).to.be.above(0);
    });

    it('Google Charts should render a pie chart', () => {
        // Ваш код Google Charts для створення графіку
        // Перевірте, чи графік правильно відображено
        // Наприклад, перевірте, чи є круговий графік у контейнері
        const googleChartContainer = document.getElementById('googleChartContainer');
        const googleChartSvg = googleChartContainer.querySelector('svg');
        expect(googleChartSvg).to.exist;
    });
});
```

Додаток Д 2

```
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Data Visualization Testing</title>

    <!-- D3.js -->
    <script src="https://d3js.org/d3.v6.min.js"></script>

    <!-- Chart.js -->
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

    <!-- Google Charts -->
    <script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
    <script type="text/javascript">
        google.charts.load('current', {packages: ['corechart']});
    </script>
</head>
<body>

    <!-- D3.js Chart Container -->
    <div id="d3ChartContainer" style="width: 400px; height: 300px;"></div>
```

```
    <!-- Chart.js Chart Container -->
    <canvas id="chartjsCanvas" width="400" height="300"></canvas>

    <!-- Google Charts Container -->
    <div id="googleChartContainer" style="width: 400px; height: 300px;"></div>

    <!-- Include your testing script -->
    <script src="test.js"></script>
</body>
</html>
```