МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Мультиплатформенний оглядач файлів з розширеним функціоналом**

Виконав: студент 4 курсу, групи 6.1211-ПЗС
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
А. В. Волович
(ініціали та прізвище)
Керівник асистент кафедри ЕІСПЗ. О. О. Жуков
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дискус»
Р.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____перший (бакалаврський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Програмне забезпечення систем____
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Т. В. Критська
"_01_" __березня_____2025_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Воловичу Андрію Володмировичу
(прізвище, ім'я, по батькові)

1.Тема роботи Мультиплатформенний оглядач файлів з розширеним функціоналом
керівник роботи _____Жуков Олександр Олексійович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 27.12.2024 № 2029-с _____
2. Строк подання студентом кваліфікаційної роботи_____02.06.2025_____
3. Вихідні дані бакалаврської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми створення мобільних застосунків;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- тестування програмної системи та розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
_____слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата
		Завдання прийняв

7. Дата видачі завдання 01.03.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Аналіз предметної області	01.03-12.03.25	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	12.03-14.03.25	виконано
3	Аналіз існуючих методів рішення	14.03-15.03.25	виконано
4	Аналіз методів розробки застосунків	16.03-24.03.25	виконано
5	Аналіз засобів розробки застосунків	25.03-30.03.25	виконано
6	Узгодження подальших дій з науковим керівником	01.04-04.04.25	виконано
7	Проєктування системи застосунка	04.03-07.04.25	виконано
8	Програмна реалізація застосунка	07.04-17.04.25	виконано
9	Представлення отриманих результатів науковому керівнику та узгодження плану подальшого дослідження	18.04-20.04.25	виконано
10	Реалізація інтерфейсу для застосунку	21.04-02.05.25	виконано
11	Тестування застосунку	02.05-07.05.25	виконано
12	Оформлення звіту	08.05-13.05.25	виконано
13	Оформлення презентації.	15.05-20.05.25	виконано

Студент _____ А. В. Волович
(підпис) (прізвище та ініціали)

Керівник роботи _____ О. О. Жуков
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок - 72

Рисунки - 18

Таблиць - 2

Джерела – 9

Волович.А.В. Мультиплатформенний оглядач файлів з розширеним функціоналом: кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник О. О. Жуков. Запоріжжя ЗНУ, 2025. 72 с.

Метою даного проєкту є розробка додатку Мультиплатформенний file explorer. Проєкт побудований з використанням Eletron та React який забезпечує зручний доступ до файлів і папок на різних операційних системах.

У процесі розробки була розглянута проблема створення оглядача файлів, який би працював на різних платформах, таких як Windows, macOS та Linux, і забезпечував би користувачам можливість легко навігувати по файловій системі, використовувати теги для організації файлів, а також мати доступ до основних тек, таких як Documents, Downloads та Мій комп’ютер, незалежно від операційної системи. Для досягнення крос-платформенності було використано Eletron. У результаті був розроблений файловий експлорер, який дозволяє користувачам переглядати файли та папки, створювати, видаляти їх, а також призначати теги для кращої організації. Експлорер має інтуїтивно зрозумілий інтерфейс користувача, який адаптується до стилю кожної операційної системи.

Ключіві слова : ОС, Eletron, React, Node.js, Windows, Linux, MacOS, інтерфейс, теги, файли, папки, системи, дані

ABSTRACT

Pages - 72

Figures - 18

Tables - 2

Sources - 9

Volovych, A.V. Multi-platform file viewer with advanced functionality bachelor's thesis in Software Engineering (specialty 121) / scientific advisor O.O. Zhukov. Zaporizhzhia: ZNU, 2025. 72 p.

The goal of this project is to develop a multi-platform file explorer application. The project is built using Electron and React, which provides convenient access to files and folders on different operating systems.

During development, we considered the problem of creating a file explorer that would work on different platforms, such as Windows, macOS, and Linux, and would allow users to easily navigate the file system, use tags to organize files, and access basic folders, such as Documents, Downloads, and My Computer, regardless of the operating system. Eletron was used to achieve cross-platform compatibility. The result was a file explorer that allows users to view files and folders, create and delete them, and assign tags for better organization. The explorer has an intuitive user interface that adapts to the style of each operating system.

Key wordЖs OS, Eletron, React, Node.js, Windows, Linux, MacOS, interface, tags, files, folders, systems, data

ЗМІСТ

ВСТУП 7

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 11

1.1 Огляд літературних джерел 11

1.2 Аналіз програмного забезпечення 11

1.3 Огляд існуючих рішень 14

1.3.1 Explorer(win10-11) 15

1.3.2 Nautilus(linux) 17

1.3.3 Finder(macos) 18

1.4 Постановка завдання 19

1.5 Висновок до розділу 1 20

2. ОГЛЯД ТА АНАЛІЗ ДОДАТКУ 21

2.1 Аналіз особливостей цільових операційних систем 21

2.2 Обґрунтування вибору інструментів розробки 23

2.3 Висновок до розділу 2 25

3. РОЗРОБКА ДОДАТКУ 27

3.1 Опис предметної області 27

3.2 Архітектура додатка 28

3.3 Функціональні та інтерфейсні вимоги 31

3.4 Реалізація 33

3.5 Тестування 66

ВИСНОВКИ 70

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 72

ВСТУП

Актуальність теми

У сучасному світі ми постійно опиняємося в справжньому вирі цифрових пристроїв. Вранці ми працюємо за офісним комп'ютером на Windows, вдень відповідаємо на листи з MacBook, а ввечері, можливо, експериментуємо з домашнім сервером на Linux. І в цій постійній зміні декорацій є одна дрібниця, яка непомітно, але впевнено псує нам життя — це робота з файлами. Кожна операційна система пропонує свій власний погляд на те, як це має відбуватися: тут у нас Провідник, там — елегантний Finder, а десь іще — функціональний Nautilus. Хоча кожен із них по-своєму хороший, вони змушують нас щоразу перелаштовуватися, створюючи відчуття фрагментованого, розірваного досвіду.

Але труднощі не закінчуються на відмінностях в інтерфейсах. Справжній головний біль починається, коли ми намагаємося навести лад у своєму цифровому господарстві. Уявіть, що на macOS ви звикли до зручної системи міток: позначили всі файли, що стосуються важливого проєкту, червоним, а особисті фото — синім, і ваш хаос перетворився на впорядковану структуру. Та варто вам перейти на інший пристрій, як уся ця магія зникає, залишаючи вас наодинці з нескінченними списками файлів. Для людей, чия робота пов’язана з величезними обсягами інформації — чи то творці контенту, програмісти, чи дослідники — така відсутність єдиної системи організації стає справжньою перешкодою на шляху до продуктивності.

Саме на цьому тлі вимальовується ідея, що здатна все змінити. Уявіть собі єдиний файловий оглядач, який працює однаково бездоганно на будь-якій платформі, будь то Windows, macOS чи Linux. Сучасні технології, такі як Electron та React, дають можливість втілити цю ідею в життя, створивши не просто програму, а надійного помічника. Такий інструмент дозволить забути про платформні відмінності та запропонує абсолютно новий рівень комфорту.

Вам подарує універсальну систему міток, яка працюватиме скрізь, дозволяючи нарешті організувати файли так, як зручно саме вам, а не так, як диктує система. Більше не доведеться щоразу шукати ключові папки, адже програма сама розумітиме, де лежать ваші «Документи», «Завантаження» чи файли «Робочого столу». Це буде єдиний простір, де інтелектуальний пошук та інтуїтивна навігація стануть нормою, а не винятком. Таким чином, створення такого універсального рішення — це не просто технічне завдання, а важливий крок до гармонії у нашій взаємодії з цифровим світом, що допоможе приборкати хаос та зробити роботу з файлами по-справжньому простою та приємною на будь-якому пристрої.

Мета дослідження

Мета цієї роботи створити файловий оглядач на базі Electron і React, який однаково працюватиме на різних ОС, підтримуватиме мітки, системні папки та зручну навігацію.

Завдання дослідження

Вивчити тему, включно з оглядом наявних файлових менеджерів на популярних системах (Windows Explorer, Nautilus, Finder).

1. Розглянути можливості Electron і React для створення швидких кросплатформних додатків.
2. Побудувати архітектуру, яка буде зручна в підтримці, не залежатиме від ОС і легко розширюватиметься.
3. Реалізувати функції.
   - 3.1. Відображення структури файлів.
   - 3.2. Підтримку міток.
   - 3.3. Збереження та відновлення міток.
   - 3.4. Виявлення системних папок (Documents, Downloads, Desktop та ін.) .
   - 3.5. Кросплатформеність за допомогою Electron.

4. Провести тестування на Windows, macOS і Linux.
5. Оцінити продуктивність, зручність інтерфейсу та можливості розвитку.
6. Підготувати документацію до програми.

Об’єкт дослідження

Об'єкт дослідження способи роботи з файлами в настільних ОС (Windows, macOS, Linux) та інтерфейси, через які користувач керує файлами.

Предмет дослідження

Предмет дослідження технології та підходи до створення кросплатформених додатків для роботи з файлами, включно з реалізацією міток і збереження метаданих через інтерфейс.

Методи дослідження

У дослідженні застосовувалися:

- Аналіз та узагальнення джерел щодо файлових систем та інтерфейсів.
- Порівняння наявних рішень.
- Програмування з використанням Electron і React.
- Модульне та інтеграційне тестування.
- Перевірка роботи програми на різних операційних системах.
- Збір зворотного зв'язку для поліпшення проєкту.

Практичне значення одержаних результатів

Результати роботи мають як теоретичну, так і прикладну користь. На практиці було створено файловий менеджер, який однаково працює в Windows, macOS і Linux, і вирішує проблему поширених завдань:

- Дає єдиний досвід незалежно від ОС.
- Дає змогу використовувати мітки - функцію, яку рідко можна зустріти на інших платформах.

- Автоматично підключає основні користувацькі папки без ручних налаштувань.
- Дає гнучку архітектуру, яку можна доповнювати, наприклад, додаванням попереднього перегляду зображень або підключенням хмарного сховища.

Програма може бути корисною і в навчальному середовищі як приклад сучасного кросплатформного застосунку, зробленого за допомогою актуального набору інструментів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд літературних джерел

Файловий менеджер (англ. file manager) — це системний програмний засіб, що надає користувачеві графічний інтерфейс для взаємодії з файловою системою комп’ютера, дозволяючи виконувати операції з файлами та директоріями. Історично ці інструменти пройшли значний еволюційний шлях: від ранніх текстових інтерфейсів командного рядка та псевдографічних оболонок, як-от Norton Commander у 1980-х роках, до сучасних, глибоко інтегрованих у середовище ОС графічних рішень, таких як Провідник (Explorer) у Windows, Finder у macOS чи Nautilus у Linux.

Цей еволюційний процес безпосередньо відображає зростаючі вимоги користувачів до ергономіки та функціональних можливостей. Сьогодні файлові менеджери є невід’ємним компонентом будь-якої сучасної операційної системи, що забезпечує інтуїтивно зрозумілий доступ до даних. Фундаментальні праці з теорії операційних систем, як-от «Operating System Concepts» А. Сільбершаца, а також дослідження в галузі проєктування інтерфейсів користувача (UI/UX) наголошують на ключовій ролі цих інструментів у забезпеченні ефективної та інтуїтивної взаємодії людини з комп’ютером.

1.2 Аналіз програмного забезпечення

Сучасні file explorers базуються на технологіях, що дозволяють інтегруватися з операційними системами та надавати широкий спектр функцій. Наприклад:

- *Провідник (Windows)* (див Рис 1) використовує WinAPI для взаємодії з файловою системою, підтримуючи базові операції (копіювання,

переміщення, видалення) та розширені функції, як пошук і попередній перегляд файлів.

- *Nautilus (Linux)* (див Рис 2) розроблений для середовища GNOME, застосовує бібліотеку GTK і має відкритий код, що дозволяє розширювати його функціональність через модулі.
- *Finder (macOS)* (див Рис 3) побудований на фреймворку Cocoa, що забезпечує глибоку інтеграцію з екосистемою Apple, наприклад, із хмарним сховищем iCloud.

Таким чином, аналіз цих рішень наочно демонструє, як технології операційної системи формують функціональні можливості файлових менеджерів. Водночас він висвітлює їхній фундаментальний недолік: жорстку прив’язку до конкретної платформи, що створює бар'єри для уніфікованого користувацького досвіду в гетерогенному середовищі.

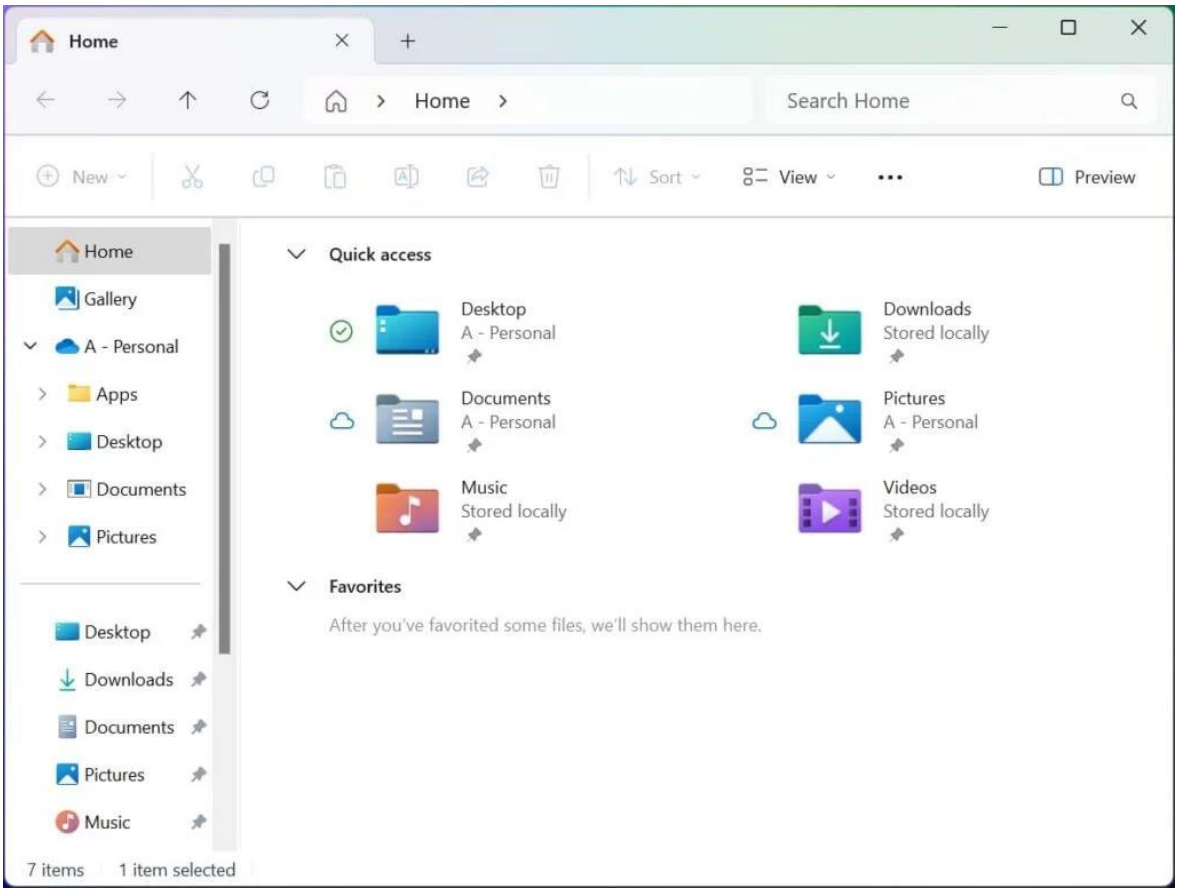


Рисунок 1 — *Explorer (win11)*

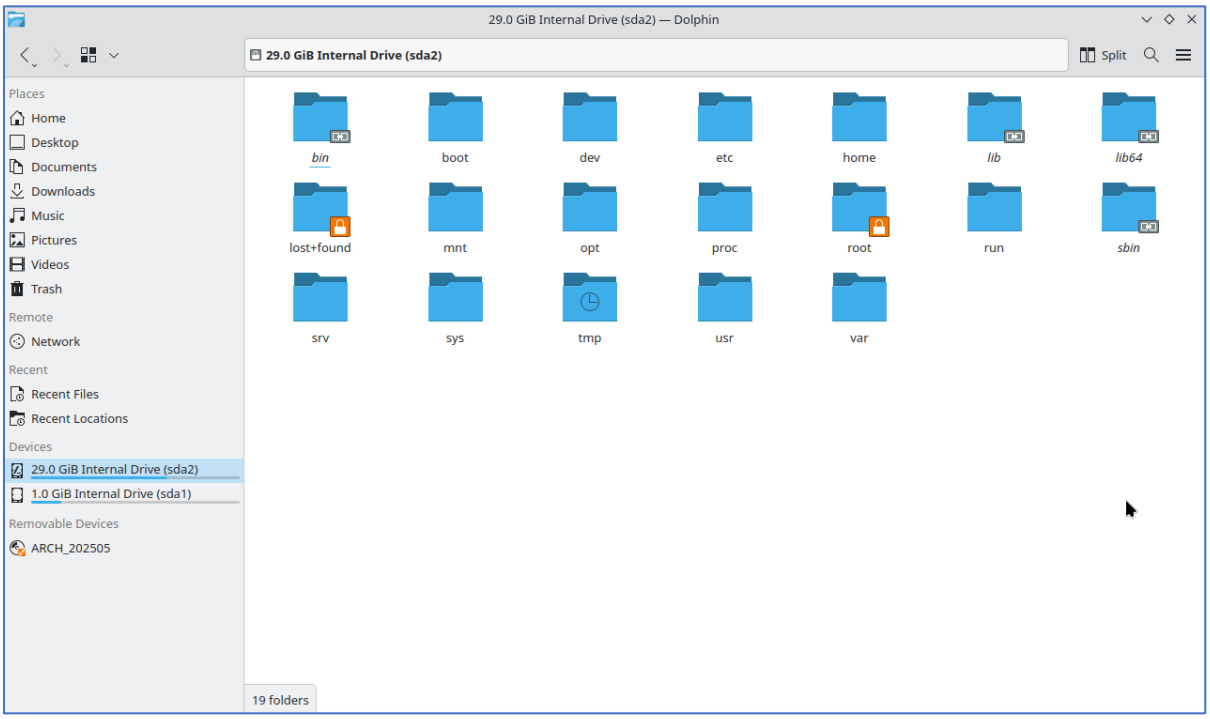


Рисунок 2 — *Nautilus*

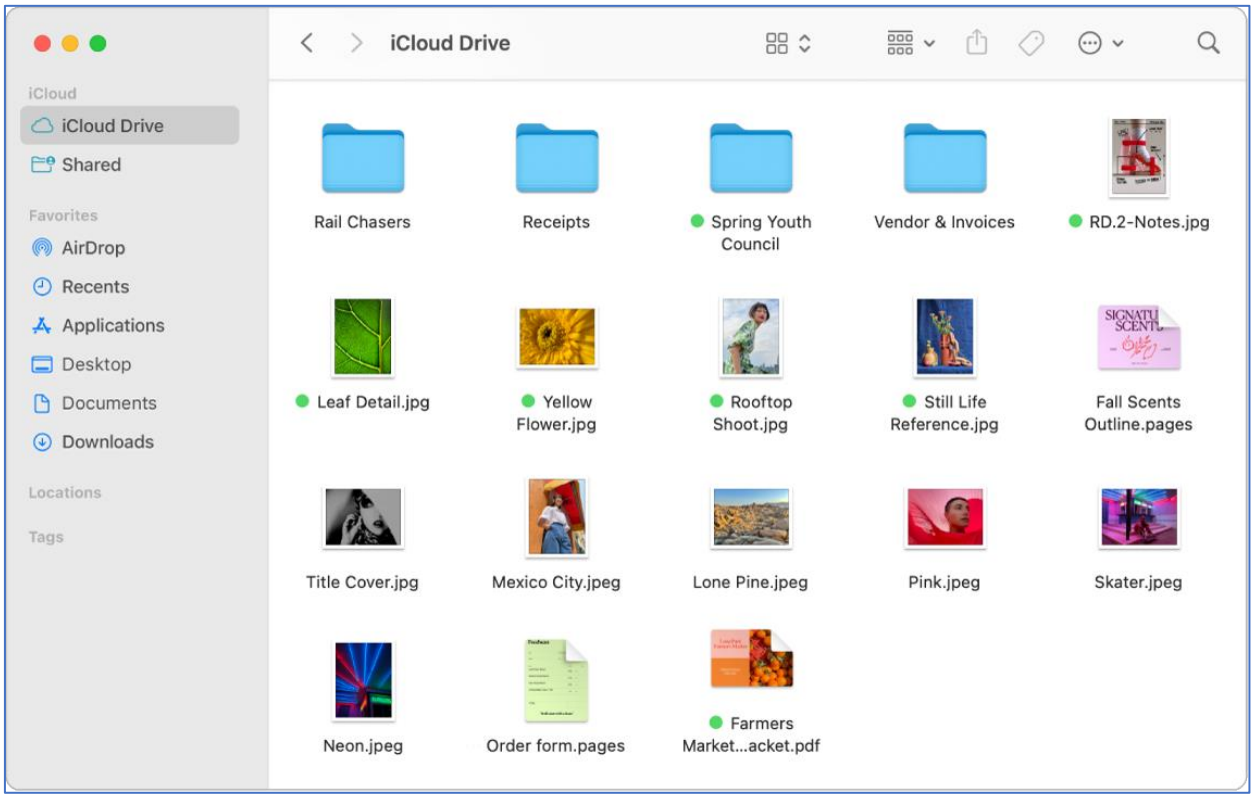


Рисунок 2 — *Finder*

1.3 Огляд існуючих рішень

Для глибшого розуміння переваг та недоліків існуючих рішень у цьому підрозділі проведено порівняльний аналіз трьох провідних файлових менеджерів: Провідника (Windows), Nautilus (Linux) та Finder (macOS). Метою аналізу є виявлення їхніх ключових функціональних характеристик, що дозволить обґрунтувати доцільність розробки нового кросплатформного рішення, орієнтованого на усунення виявлених недоліків.

Для об'єктивного порівняння було обрано низку критеріїв, що відображають як технічні можливості програм, так і очікування сучасних користувачів. До цих критеріїв належать:

Кросплатформність — можливість функціонування в різних операційних системах, що є фундаментальною вимогою для універсального інструменту.

Підтримка тегів (міток) — наявність вбудованої системи тегування файлів для їх гнучкої організації та каталогізації.

Доступ до системних директорій — рівень інтеграції та зручність доступу до стандартних користувацьких папок (наприклад, «Документи», «Завантаження»).

Інтерфейс користувача (UI) — загальна ергономічність, візуальна привабливість та інтуїтивність інтерфейсу.

Розширюваність — можливість модифікації та розширення стандартної функціональності за допомогою плагінів або інших механізмів.

Продуктивність — швидкодія під час виконання основних файлових операцій та навігації по директоріях.

Результати порівняльного аналізу за зазначеними критеріями для Windows Explorer, Nautilus та macOS Finder зведено у Таблицю 1

Таблиця 1 — *Порівняльний аналіз трьох популярних файлових менеджерів*

Критерій	Explorer	Finder	Naulilus
Кросплатформність	Windows	Linux	macOS
Підтримка тегів	Відсутня	Відсутня	вбудована функція
Доступ до основних тек	інтегрований у систему	налаштовується	інтегрований у систему
Інтерфейс користувача	класичний, з вкладками у Windows 11	залежить від середовища	інтуїтивний, з жестами
Розширюваність	обмежені можливості	відкритий код, плагіни	обмежені можливості
Продуктивність	оптимізований для Windows	залежить від системи	оптимізований для macOS

1.3.1 Explorer (win10-11)

Провідник (Explorer) є штатним файловим менеджером, інтегрованим в операційні системи сімейства Windows. Він виступає основним інструментом для взаємодії користувача з файловою системою, забезпечуючи доступ до файлів, директорій та накопичувачів. Провідник глибоко інтегрований із системними функціями, такими як хмарне сховище OneDrive, та реалізує повний набір стандартних файлових операцій.

Ключові характеристики:

- *Кросплатформність*. Відсутня. Провідник є ексклюзивним компонентом ОС Windows і не підтримується на інших платформах, таких як macOS або Linux.

- *Підтримка тегів.* Нативна функціональність для тегування файлів відсутня, що значно обмежує можливості їх семантичної організації та каталогізації.
- *Доступ до системних директорій.* Високий. Забезпечується швидкий доступ до стандартних користувацьких папок (Documents, Downloads, Desktop) та дискових накопичувачів через вузол «Цей ПК».
- *Інтерфейс користувача.* Зберігає класичний підхід, який є звичним для мільйонів користувачів. У Windows 11 інтерфейс було модернізовано, зокрема додано підтримку вкладок, однак у порівнянні з сучасними UI-трендами він може сприйматися як дещо консервативний.
- *Розширюваність.* Обмежена. Можливості кастомізації та інтеграції сторонніх плагінів є мінімальними, хоча підтримується розширення контекстного меню сторонніми програмами.
- *Продуктивність.* Висока. Як нативний компонент, Провідник оптимізований для роботи з файловою системою NTFS і демонструє високу швидкодію навіть при роботі з великими масивами даних.

Переваги:

- Глибока інтеграція з ОС Windows, що гарантує стабільність та високу продуктивність.
- Безшовна інтеграція з хмарним сервісом OneDrive.
- Наявність вкладок у версії для Windows 11.

Недоліки:

- Відсутність нативної системи тегування, що ускладнює семантичну організацію даних.
- Обмежені можливості кастомізації інтерфейсу та розширення функціоналу.
- Платформна залежність, що унеможливлює використання в гетерогенних середовищах.

1.3.2 Nautilus (linux)

Nautilus, також відомий як GNOME Files, є стандартним файловим менеджером для популярного робочого середовища GNOME у дистрибутивах Linux. Завдяки відкритому вихідному коду, він є гнучким інструментом, орієнтованим на простоту інтерфейсу та тісну інтеграцію з екосистемою GNOME.

Ключові характеристики:

- *Кросплатформність*. Обмежена. Nautilus призначений для функціонування в середовищі Linux і не підтримується в ОС Windows або macOS.
- *Підтримка тегів.* Нативна підтримка тегів відсутня. Частково ця функціональність може бути реалізована за допомогою сторонніх розширень, що потребує додаткових налаштувань.
- *Доступ до системних директорій*. Високий. Бічна панель забезпечує швидкий доступ до домашньої директорії, стандартних папок та змонтованих накопичувачів.
- *Інтерфейс користувача.* Сучасний та мінімалістичний, відповідає гайдлайнам GNOME. Зовнішній вигляд залежить від обраної системної теми, що забезпечує візуальну узгодженість.
- *Розширюваність.* Висока. Відкритий вихідний код та підтримка плагінів дозволяють значно розширювати базову функціональність, наприклад, додавати інтеграцію з хмарними сервісами чи інструментами розробника
- *Продуктивність.* Середня. Швидкодія може варіюватися залежно від конфігурації системи та дистрибутиву Linux. При роботі з директоріями, що містять велику кількість файлів, іноді спостерігаються затримки.

Переваги:

- Відкритий вихідний код і висока розширюваність.

- Гнучка конфігурація бічної панелі для швидкого доступу.
- Тісна інтеграція з сервісами та застосунками робочого середовища GNOME.

Недоліки:

- Відсутність вбудованої системи тегування файлів.
- Жорстка прив'язка до робочого середовища GNOME.
- Повна відсутність підтримки інших операційних систем.

1.3.3 Finder (macos)

Finder є невід'ємним компонентом операційної системи macOS, що виконує роль файлового менеджера та графічної оболонки. Він розроблений компанією Apple з акцентом на інтуїтивність, естетику та глибоку інтеграцію з апаратною та програмною екосистемою.

Ключові характеристики:

- *Кросплатформність.* Відсутня. Finder є ексклюзивним програмним забезпеченням для macOS.
- *Підтримка тегів.* Висока. Нативна підтримка тегів є однією з ключових переваг. Система дозволяє присвоювати файлам кольорові мітки та ключові слова, які інтегровані в системний пошук Spotlight.
- *Доступ до системних директорій*. Високий. Бічна панель надає миттєвий доступ до стандартних папок, пристроїв та хмарного сховища iCloud Drive.
- *Інтерфейс користувача*. Відповідає високим стандартам Apple щодо ергономіки та дизайну, є інтуїтивно зрозумілим і підтримує розширені можливості взаємодії, такі як жести Multi-Touch.
- *Розширюваність*. Низька. Можливості для глибокої кастомізації чи додавання сторонніх плагінів сильно обмежені.

- Продуктивність. Висока. Finder оптимізований для роботи з файловою системою APFS та апаратним забезпеченням Apple, що забезпечує високу швидкість роботи.

Переваги:

- Потужна вбудована система тегування для організації файлів.
- Високоякісний, інтуїтивний та естетично привабливий інтерфейс.
- Безшовна інтеграція з iCloud та іншими сервісами екосистеми Apple.

Недоліки:

- Повна відсутність кросплатформності.
- Обмежені можливості розширення функціоналу.
- Замкненість в екосистемі Apple.

1.4 Постановка завдання

На основі проведеного аналізу, який виявив ключові недоліки існуючих рішень (зокрема, відсутність кросплатформності та уніфікованих засобів організації даних), формулюється постановка завдання для даної дипломної роботи.

Метою є розробка кросплатформного файлового менеджера, який відповідає таким ключовим вимогам:

- *Універсальність*. Забезпечення стабільної роботи та ідентичного користувацького досвіду на операційних системах Windows, macOS та Linux.
- *Розширений функціонал*. Реалізація універсальної системи тегування (присвоєння міток) для гнучкої каталогізації файлів незалежно від платформи.
- *Зручність*. надання уніфікованого та автоматизованого доступу до стандартних системних директорій («Документи», «Завантаження» тощо).

Такий підхід дозволить створити інструмент, що задовольняє сучасні потреби користувачів, які працюють у гетерогенному цифровому середовищі.

1.5 Висновок до розділу 1

Таким чином, проведений у першому розділі аналіз предметної області продемонстрував, що, незважаючи на високий рівень розвитку сучасних файлових менеджерів, жоден із них не пропонує універсального рішення для багатоплатформного використання.

Провідник (Windows) демонструє високу продуктивність, але є функціонально обмеженим та нерозширюваним. Nautilus (Linux) пропонує гнучкість завдяки відкритому коду, однак позбавлений нативної підтримки тегів і прив'язаний до середовища GNOME. Finder (macOS) вирізняється якісним інтерфейсом та системою тегування, але є повністю замкненим у своїй екосистемі.

Отже, розробка універсального кросплатформного файлового менеджера з розширеним функціоналом, зокрема з підтримкою тегів, є актуальним та практично значущим завданням, спрямованим на подолання існуючої фрагментації користувацького досвіду.

2 ОГЛЯД ТА АНАЛІЗ ДОДАТКУ

а. Аналіз особливостей цільових операційних систем

Для розробки ефективного кросплатформного файлового менеджера необхідно враховувати фундаментальні відмінності в архітектурі файлових систем та організації даних у цільових операційних системах.

Основою зберігання даних у сучасних версіях ОС Windows є файлова система NTFS (New Technology File System), розроблена корпорацією Microsoft. Вона стала стандартом, починаючи з Windows NT 3.1, завдяки низці *ключових переваг:*

- *Журналювання (Journaling).* NTFS використовує механізм журналювання транзакцій, за якого всі зміни спершу записуються в спеціальний лог-файл (журнал). Це забезпечує високу відмовостійкість: у разі збою живлення система може відновити свою цілісність, мінімізуючи ризик втрати даних.
- *Вбудоване стиснення та шифрування.* NTFS підтримує прозоре стиснення файлів і директорій на основі алгоритму LZ77, а також шифрування на рівні файлової системи (Encrypting File System, EFS), що дозволяє захищати конфіденційні дані.
- *Розширена система прав доступу (ACL).* Механізм списків контролю доступу (Access Control Lists) дозволяє гнучко налаштовувати права доступу для окремих користувачів та груп.
- *Підтримка великих обсягів даних.* NTFS теоретично підтримує розділи об'ємом до 16 КБ та файли розміром до 256 ТБ.
- *Альтернативні потоки даних (ADS).*Файли в NTFS можуть містити додаткові, приховані потоки даних, що використовуються для зберігання метаданих.

На відміну від Windows, у Linux-системах організація каталогів підпорядковується стандарту FHS (Filesystem Hierarchy Standard). FHS

визначає чітку та логічну структуру директорій для Unix-подібних систем, зображену на (див Рис. 4)

Ключові принципи FHS:

- *Чіткий поділ.* Системні файли, користувацькі дані та конфігураційні файли зберігаються в окремих, стандартизованих директоріях, що спрощує адміністрування, резервне копіювання та оновлення системи.
- *Гнучкість файлових систем.* Структура FHS є незалежною від конкретної файлової системи, дозволяючи використовувати різні типи (ext4, Btrfs, XFS) для різних розділів.
- *Універсальність.* Стандарт адаптований як для потужних серверів, так і для настільних комп'ютерів та вбудованих систем.

Недоліком для нових користувачів може бути складність запам'ятовування призначення кожної директорії.

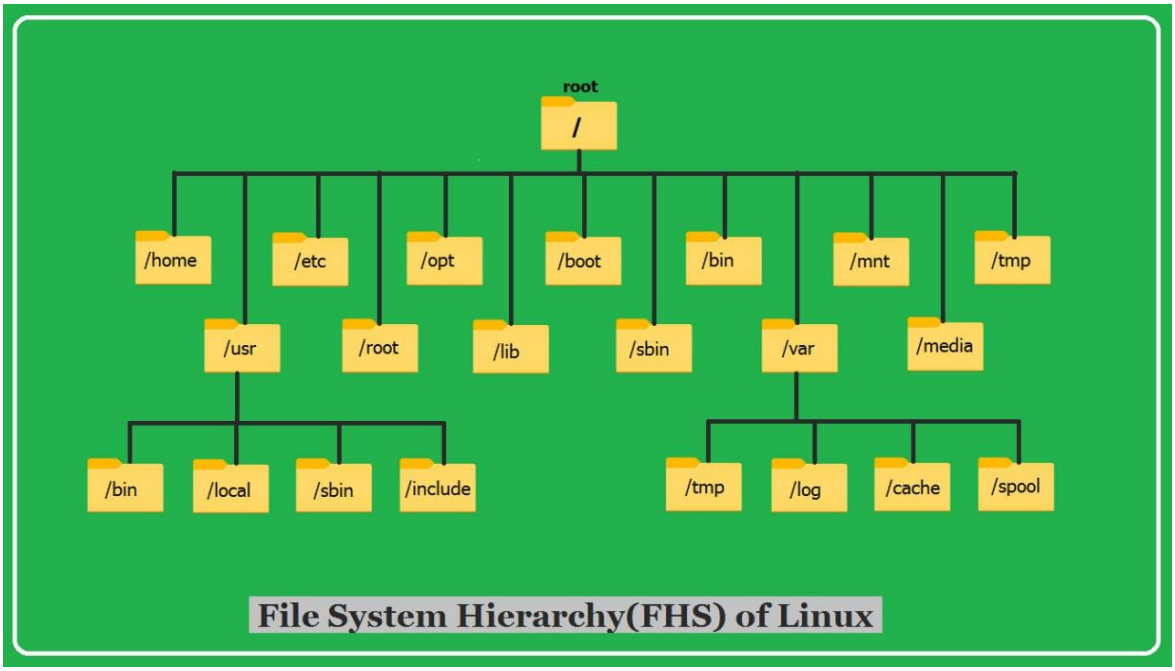


Рисунок 4 — *FHS Linux*

Важливим аспектом кросплатформності є підтримка різних апаратних архітектур. Найпоширенішими сьогодні є x86-64 та ARM.

x86-64 (AMD64). 64-бітна архітектура, що домінує на ринку настільних комп'ютерів, ноутбуків та серверів. Вона характеризується високою продуктивністю та широкою сумісністю з програмним забезпеченням.

ARM. Архітектура з набором скорочених команд (RISC), що є стандартом для мобільних пристроїв (смартфонів, планшетів) та вбудованих систем завдяки високій енергоефективності. Останнім часом архітектура ARM активно використовується і в ноутбуках (наприклад, Apple Silicon).

Обидві ОС — Windows і Linux — мають версії, адаптовані для роботи як на x86-64, так і на ARM. Розуміння цих відмінностей є критично важливим для забезпечення справжньої кросплатформності розроблюваного застосунку.

b. Обґрунтування вибору інструментів розробки

Electron — це фреймворк для створення настільних додатків з використанням веб-технологій, таких як HTML, CSS [1] і JavaScript. Він базується на Chromium і Node.js, що дозволяє розробникам використовувати всі можливості цих платформ [2].

Основні переваги Electron:

- *Кросплатформність*. Додатки, створені на Electron, можуть працювати на Windows, macOS і Linux без необхідності значних змін у коді.*Доступ до файлової системи.* Завдяки інтеграції з Node.js, Electron надає повний доступ до файлової системи через модуль fs, що є критичним для file explorer.
- *Використання веб-технологій.* Розробники можуть використовувати знайомі інструменти та бібліотеки, що прискорює процес розробки.

Electron ідеально підходить для проєкту file explorer, оскільки забезпечує єдину кодову базу для всіх платформ і дозволяє реалізувати складні функції, такі як робота з файлами та тегами.

React — це бібліотека для створення інтерактивних інтерфейсів користувача. Вона дозволяє будувати UI за допомогою компонентів, що робить код більш структурованим і легким для підтримки [4].

Основні переваги React:

- *Компонентний підхід.* Інтерфейс розбивається на незалежні компоненти, що спрощує розробку та тестування.
- *Віртуальний DOM.* Підвищує продуктивність, особливо при роботі з великими списками файлів.
- *Екосистема.* React має велику кількість бібліотек і інструментів для розширення функціональності.

У file explorer React дозволяє створити зручний і чуйний інтерфейс, який легко адаптується до різних розмірів вікон і роздільної здатності екранів.

Для зберігання тегів файлів обрано формат JSON. JSON (JavaScript Object Notation) — це легкий формат обміну даними, який легко читається і записується як людьми, так і машинами.

Переваги використання JSON для тегів:

- *Простота*. JSON є простим у використанні та інтеграції з JavaScript, що є основною мовою для Electron і React.
- *Гнучкість.* Дозволяє легко додавати, видаляти та оновлювати теги без складних запитів до бази даних.
- *Легкість доступу.* Файли JSON можуть бути збережені локально, що забезпечує швидкий доступ до даних без необхідності підключення до зовнішньої бази даних.

У контексті file explorer, JSON дозволяє зберігати теги для файлів у структурованому вигляді, що полегшує їхнє відображення та фільтрацію в інтерфейсі користувача.

blueprintjs — це набір компонентів для React, який спрощує створення складних і красивих інтерфейсів користувача.

Основні переваги blueprintjs:

- *Готові компоненти.* Надає широкий вибір компонентів, таких як кнопки, меню, таблиці, що прискорює розробку.
- *Стилізація*. Компоненти мають сучасний дизайн і легко налаштовуються під потреби додатка.
- *Сумісність з React.* Легко інтегрується з React, що дозволяє швидко створювати складні UI.

У file explorer blueprintjs використовується для створення інтуїтивного інтерфейсу, що включає навігацію, списки файлів, панелі інструментів тощо.

node-disk-info — це бібліотека для Node.js, яка дозволяє отримувати інформацію про диски та файлові системи.

Основні переваги node-disk-info:

- *Кросплатформність.* Працює на Windows, macOS і Linux, що відповідає вимогам мультиплатформності.
- *Інформація про диски.* Надає дані про доступні диски, їхній об'єм, вільне місце тощо.
- *Легкість використання.* Просте API для отримання необхідної інформації.

У file explorer, node-disk-info використовується для відображення інформації про диски, що дозволяє користувачам бачити доступні диски.

с. Висновок до розділу 2

Проведений у другому розділі аналіз технологій підтвердив, що обраний стек інструментів є оптимальним для розробки кросплатформного файлового менеджера. Фреймворк Electron забезпечує фундаментальну кросплатформність та доступ до нативних функцій системи, тоді як бібліотека React дозволяє створити сучасний, продуктивний та легко підтримуваний інтерфейс користувача. Використання формату JSON для зберігання метаданих є простим та ефективним рішенням, а UI-бібліотека Blueprint.js та системна бібліотека node-disk-info значно прискорюють розробку та

розширюють функціональні можливості застосунку. Таким чином, обрані інструменти створюють синергію, що дозволяє реалізувати всі поставлені перед проєктом завдання.

3 РОЗРОБКА ДОДАТКУ

а. Опис предметної області

Предметна область даного дослідження охоплює процеси проєктування та розробки кросплатформного файлового менеджера з розширеним функціоналом. Застосунок призначений для забезпечення зручної навігації, організації та управління файлами в операційних системах Windows, macOS та Linux. Ключовими аспектами предметної області є: навігація по файловій системі, реалізація системи тегування (аналогічної до macOS), файловий пошук, базові операції з файлами (створення, видалення, копіювання, переміщення), а також створення сучасного інтерфейсу з підтримкою багатопанельного режиму та темної теми.

Метою розробки є створення повноцінного настільного застосунку, що дозволяє користувачам ефективно взаємодіяти з файловою системою, організовувати дані за допомогою тегів та виконувати файлові операції через інтуїтивно зрозумілий інтерфейс. Застосунок має бути кросплатформним, стабільним та забезпечувати надійну роботу з даними.Ця робота зосереджена на:

- Створенні зручного та інтуїтивного інтерфейсу для навігації по файловій системі.
- Реалізації функціоналу тегів для організації файлів.
- Забезпеченні базових операцій з файлами (створення, видалення, копіювання, переміщення).
- Розробці інтерфейсу з підтримкою розділення екрана та темної теми.
- Інтеграції з файловою системою через Electron та Node.js.

  У цій роботі буде розглянуто:

- Архітектуру та дизайн десктопного додатку з використанням Electron та React.
- Вибір та використання технологій: Electron, React, TypeScript, Node.js.

- Процес інтеграції з файловою системою різних ОС.
- Забезпечення безпеки та продуктивності додатку.
- Тестування функціональності та зручності використання.

Цей проєкт має на меті не лише створення функціонального файлового експлорера, але й демонстрацію можливостей сучасних технологій для розробки крос-платформених десктопних додатків, що покращують користувацький досвід та продуктивність.

b. Архітектура додатка

Архітектура file explorer побудована на основі фреймворку Electron, який розділяє додаток на два основні процеси:

- *Головний процес (Main Process).* Відповідає за взаємодію з операційною системою, роботу з файловою системою та управління вікнами додатка. Використовує модулі Node.js (fs, path, node-disk-info) для доступу до файлів і дисків, а також JSON-файл для зберігання тегів.
- *Процес рендерингу (Renderer Process).* Відповідає за відображення інтерфейсу користувача за допомогою React і бібліотеки blueprintjs. Цей процес комунікує з головним процесом через IPC (Inter-Process Communication) для отримання даних про файли, теки, теги та диски.

(див Рис 5) Схема архітектури.

- *Frontend (React + blueprintjs).* Інтерфейс користувача, що включає компоненти для навігації, відображення списків файлів, управління тегами та інформації про диски.
- *Backend (Electron + Node.js).* Логіка роботи з файловою системою, JSON-базою тегів і бібліотекою node-disk-info.
- *JSON.* Локальний файл (tags.json) для зберігання даних про теги.
- *IPC.* Механізм передачі даних між процесами для виклику функцій файлової системи та оновлення UI.

Основні компоненти архітектури

Electron дозволяє створювати десктопні додатки з використанням веб-технологій (HTML, CSS, JavaScript/TypeScript). Він комбінує два основні компоненти:

- Chromium: Для рендерингу інтерфейсу користувача (UI). Chromium є основою браузера Google Chrome і забезпечує відображення веб-сторінок у десктопному додатку.
- Node.js: Для доступу до системних ресурсів, зокрема файлової системи. Node.js дозволяє виконувати операції, які зазвичай недоступні у браузері, такі як читання та запис файлів, взаємодія з ОС тощо.Як Electron робить веб-додаток десктопним: Electron об’єднує веб-інтерфейс (Chromium) із серверною логікою (Node.js) у єдиному процесі. Завдяки цьому веб-додаток отримує доступ до нативних функцій ОС через API Electron, таких як робота з файлами, контекстні меню, системні сповіщення тощо. Це дозволяє розробляти десктопні додатки з однією кодовою базою, яка працює на Windows, macOS та Linux [6].

React використовується для побудови інтерфейсу користувача. Він надає кілька ключових переваг:

- Віртуальний DOM: React створює віртуальну копію DOM, що дозволяє оптимізувати оновлення інтерфейсу. Замість прямих маніпуляцій із реальним DOM при кожній зміні (наприклад, оновлення списку файлів), React спочатку оновлює віртуальний DOM, порівнює його з попередньою версією та застосовує лише мінімальні зміни до реального DOM. Це значно підвищує продуктивність, особливо у складних інтерфейсах із великою кількістю елементів, таких як списки файлів чи теги.
- Управління станом: React надає вбудовані механізми для управління станом компонентів, такі як хуки useState та useEffect. Наприклад, useState може зберігати поточну директорію або вибрані файли, а useEffect — реагувати на зміни стану, викликаючи оновлення UI чи

запити до файлової системи. Для складних додатків із централізованим станом (наприклад, для пошуку чи тегів) можна використовувати бібліотеки на кшталт Redux або MobX.

Node.js інтегровано в Electron і відповідає за серверну логіку та доступ до файлової системи. Завдяки асинхронній моделі Node.js швидко обробляє операції, такі як читання великих директорій чи копіювання файлів, не блокуючи основний потік виконання.

Інтеграція з файловою системоюElectron через Node.js надає API для роботи з файловою системою (наприклад, модуль fs). Це дозволяє додатку читати списки файлів, створювати папки, переміщати файли тощо. Уніфікований доступ до файлової системи забезпечує коректну роботу на різних ОС, враховуючи їхні особливості (наприклад, роздільники шляху: \ у Windows, / у Linux/macOS).

Детальна архітектура.

- Клієнтський рівень (UI):
- Інтерфейс побудовано на React із використанням HTML та CSS для структури та стилізації.
- Компоненти React відображають списки файлів, панель навігації, теги, вікно пошуку тощо.
- Віртуальний DOM оптимізує оновлення UI при змінах у файловій системі (наприклад, додавання файлу).

1. Серверний рівень (Electron + Node.js):

- Electron запускає два основні процеси: Main Process (керує додатком і системними операціями) та Renderer Process (рендеринг UI через Chromium).
- Node.js обробляє запити до файлової системи та передає дані до UI через IPC (Inter-Process Communication).

2. Управління станом:

- Локальний стан компонентів керується через useState (наприклад, поточна папка).

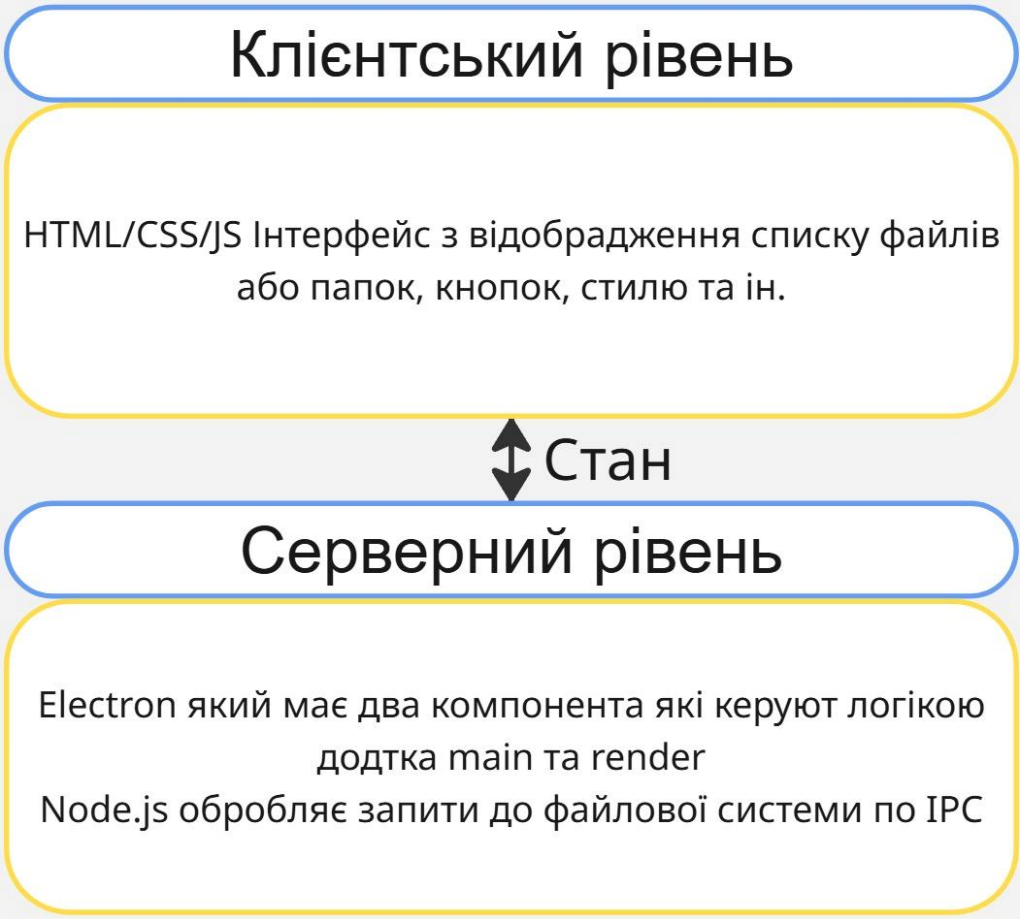


Рисунок 5 — *Архітектура застосунка*

с. Функціональні та інтерфейсні вимоги

Функціональні вимоги

Кросплатформність. Додаток має працювати однаково ефективно на Windows, macOS і Linux, забезпечуючи єдиний функціонал і інтерфейс незалежно від операційної системи.

Перегляд файлів і тек. Користувач може переглядати вміст директорій, відкривати файли, переходити між теками та повертатися назад за допомогою навігаційних елементів.

Тегування файлів. Дозволяє додавати, редагувати та видаляти теги для файлів, а також фільтрувати файли за тегами, використовуючи дані, збережені у JSON-файлі (tags.json).

Доступ до основних тек. Забезпечує швидкий доступ до стандартних директорій, таких як Documents, Downloads, та кореневих дисків (наприклад, "Мій комп'ютер" на Windows), адаптуючись до специфіки кожної платформи.

Інформація про диски. Відображає дані про доступні диски (загальний об’єм, вільне місце) за допомогою бібліотеки node-disk-info.

Продуктивність. Додаток має швидко обробляти запити до файлової системи, включаючи читання великих директорій, і відображати списки файлів без затримок, використовуючи оптимізацію через React.

Інтерфейсні вимоги

Простота використання. Інтерфейс має бути інтуїтивно зрозумілим, дозволяючи користувачам швидко виконувати ключові дії, такі як навігація по теках, додавання тегів і перегляд інформації про диски. Логічне групування елементів і чітке маркування підвищують зручність.

Естетика та привабливість. Сучасний дизайн, побудований на основі бібліотеки blueprintjs, забезпечує чисту естетику з послідовною кольоровою гамою, типографікою та іконографією, що створює відчуття надійності та професіоналізму.

Адаптивність. Інтерфейс має бути адаптивним до різних розмірів вікон і роздільних здатностей екранів, що є важливим для настільних додатків, які можуть відкриватися на моніторах із різними параметрами.

Навігація. Зрозуміла система навігації включає панель для швидкого доступу до основних тек, дисків і тегів, а також "хлібні крихти" для відображення поточного шляху в директорії.

Користувацький досвід. Інтерфейс забезпечує інтерактивний досвід завдяки використанню компонентів blueprintjs, таких як кнопки, контекстні меню та таблиці. Користувач отримує негайний зворотний зв’язок на дії, наприклад, при додаванні тегу або зміні директорії.

Доступність. Додаток відповідає стандартам доступності (WCAG 2.1), включаючи підтримку клавіатурної навігації, альтернативний текст для іконок

і сумісність зі зчитувачами екрану для користувачів із обмеженими можливостями.

Швидкість та продуктивність. Інтерфейс оптимізовано для швидкого завантаження та відображення даних, використовуючи асинхронні запити до файлової системи через fs.promises і віртуалізацію списків для великих директорій.

Відображення інформації. Дані про файли (назва, розмір, дата модифікації, теги) та диски (об’єм, вільне місце) відображаються у структурованій формі, наприклад, у вигляді таблиць або списків. Для тегів передбачено фільтри та візуальні позначки (кольорові мітки), що полегшують пошук.

d. Реалізація

У рамках підготовки проекту я спочатку налаштував середовище розробки та створив базову структуру Мультиплатформенний оглядач файлів з розширеним функціоналом, розробленого з використанням Electron та React з TypeScript. Для забезпечення якості коду та його читабельності я встановив ESLint для перевірки синтаксису TypeScript і JavaScript, а також Prettier для автоматичного форматування коду [7].

Оскільки проект базується на Electron, який включає інструменти для розробки десктопних додатків. Після завершення налаштування середовища я ініціалізував проект і встановив необхідні пакети, такі як Electron, React, TypeScript, тощо, підготувавши основу для подальшого розвитку та реалізації функціональності [3].

Налаштування середовища розробки

Для коректної роботи проекту я створив ключові файли:

- *main.ts.* Основний файл для процесу Electron, який відповідає за створення вікон додатку та взаємодію з файловою системою.

- *preload.ts.* Забезпечує безпечну комунікацію між основним і рендерерним процесами.
- *React-компоненти.* У рендерерному процесі створено основний файл для відображення інтерфейсу користувача.

Ці кроки заклали фундамент для розробки додатку з чітким розподілом між основним і рендерерним процесами, що є типовим для архітектури Electron.

Структура проекту

Проект побудовано на модульній архітектурі з використанням компонентного підходу (див Рис 6).

Основні компоненти включають:

- *FileExplorer*. Головний компонент, що координує інтерфейс файлового експлорера.
- *FilePanel*. Відповідає за відображення окремих панелей із файлами.
- *Sidebar*. Забезпечує швидкий доступ до тегів і закладок.
- *Інші компоненти*. Для відображення списків файлів, навігації та виконання операцій.

Для управління станом я використав React Hooks і Context API, що спрощує контроль над станом додатку. Для роботи з файловою системою реалізовано IPC (Inter-Process Communication) в Electron, що дозволяє рендерерному процесу надсилати запити до основного для виконання операцій, таких як читання директорій чи створення файлів.

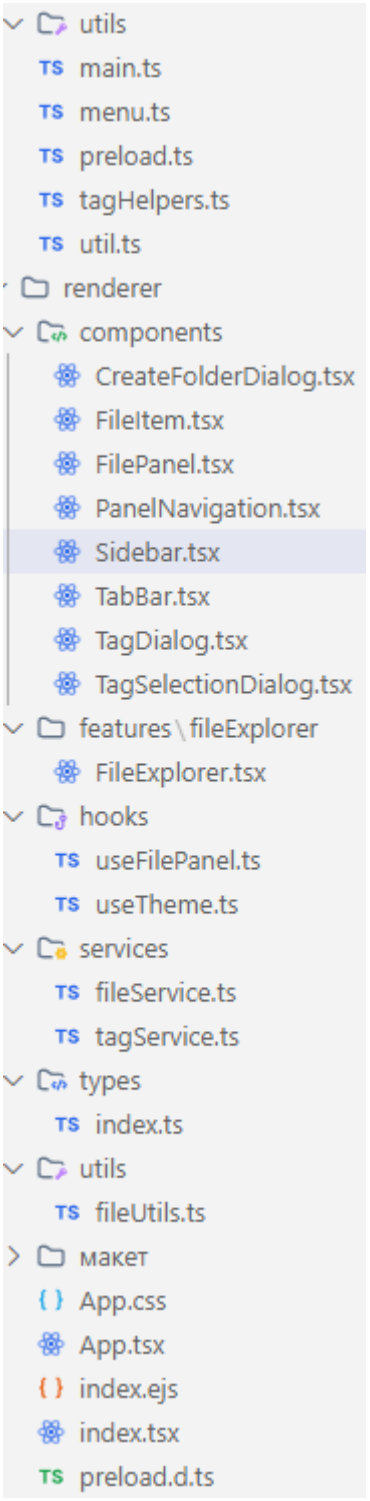


Рисунок 6 — *Структура проекту*

Реалізація основного функціоналу

Основний процес

У файлі main.ts я налаштував:

- Створення головного вікна додатку.

- Обробники IPC для взаємодії з рендерерним процесом.
- Підключення меню та інших системних функцій.

Рендерерний процес.

- У рендерерному процесі створено React-компонент FileExplorer, який включає.
- Дві панелі для навігації по файловій системі.
- Бічну панель із тегами та закладками.
- Панель інструментів для операцій із файлами.

Функціонал тегів.

- Я реалізував систему тегів, яка дозволяє.
- Додавати теги до файлів і папок.
- Фільтрувати файли за тегами.
- Керувати тегами через сервіс tagHelpers.ts.

У файлі main.ts йдут запити для взаємодії з ОС наприклад такі IPC

Лістінг 1 асинхронна функція deleteFileOrFolder, яка призначена для видалення файлів або папок за заданим шляхом (filePath). Ця функція використовує модуль fs.promises із Node.js для операцій із файлами та враховує особливості операційної системи через перевірку os.platform(). Якщо на Linux виникають проблеми з правами доступу, функція викликає окремий метод deleteFileOrFolderWithSudo для видалення з підвищеними привілеями. Давайте розберемо код покроково.

Лістінг 1 — *Видаленя "main.ts"*

```
const deleteFileOrFolder = async (filePath: string) => {
  if (os.platform() === 'linux') {
    filePath = filePath.replace(/\\/g, '/');
  }
  try {
    const stats = await fs.promises.stat(filePath);
    if (stats.isDirectory()) {
      if (typeof fs.promises.rm === 'function') {
        await fs.promises.rm(filePath, { recursive: true,
force: true });
      } else {
```

```
        await fs.promises.rmdir(filePath, { recursive: true
});
      }
    } else {
      await fs.promises.unlink(filePath);
    }
    return { success: true };
  } catch (error: any) {
    if (
      os.platform() === 'linux' &&
      (error.code === 'EACCES' || error.code === 'EPERM')
    ) {
      return await deleteFileOrFolderWithSudo(filePath);
    }
    return { success: false, error: error.message };
  }
};
```

Лістинг 2 використовує модуль ipcMain для обробки запитів від рендерер-процесу (фронтенду) до головного процесу (бекенду). Запит із назвою 'get-files' призначений для отримання списку файлів і папок у вказаному каталозі (currentPath) разом із детальною інформацією про кожен елемент, такою як ім'я, тип (файл чи папка), розмір (для файлів) і дата створення.

Лістінг 2 — *Запит файлів "main.ts"*

```
ipcMain.handle('get-files', async (_, currentPath: string)
=> {
    try {
      const entries = await
fs.promises.readdir(currentPath, {
        withFileTypes: true,
      });
      const fileEntries = await Promise.all(
        entries.map(async (entry) => {
          const filePath = path.join(currentPath,
entry.name);
          try {
            const stats = await fs.promises.stat(filePath);
            return {
              name: entry.name,
              isDirectory: entry.isDirectory(),
```

```
              size: entry.isDirectory() ? null :
stats.size,
              creationDate: stats.birthtime,
            };
          } catch (error: any) {
            if (error.code === 'ENOENT') {
              console.warn(
                `Файл не найден: ${filePath}. Возможно, он
был удалён.`,
              );
            }
            return null;
          }
        }),
      );
      return fileEntries.filter((entry) => entry !== null);
    } catch (error: any) {
      throw error;
    }
  });
```

Лістинг 3 використовує модуль ipcMain для обробки запитів від рендерер-процесу (фронтенду) до головного процесу (бекенду). Запит із назвою 'get-disks' призначений для отримання списку монтованих дисків або точок монтування, адаптованих до операційної системи, на якій запущено додаток. Код враховує особливості Windows, macOS та Linux, повертаючи відповідні дані для кожної платформи.

Лістинг 3 — *Запис дисків"main.ts"*

```
ipcMain.handle('get-disks', async () => {
    const platform = os.platform();
    try {
      const disks = await getDiskInfo();
      if (platform === 'win32') {
        return disks.map((disk) => disk.mounted);
      }
      if (platform === 'darwin') {
        return disks
          .map((disk) => disk.mounted)
          .filter((p: string) => p.startsWith('/Volumes/'))
          .map((p: string) => p.replace('/Volumes/', ''));
      }
      if (platform === 'linux') {
        return disks
```

```
            .filter((disk) => disk.mounted && disk.mounted
!== 'n/a')
            .map((disk) => disk.mounted);
        }
        return [];
      } catch (error) {
        return [];
      }
    });
```

Лістинг 4 обробляє подію IPC з іменем 'show-context-menu'. Вона спрацьовує, коли користувач, наприклад, клацає правою кнопкою миші в інтерфейсі програми. У відповідь створюється контекстне меню з діями для роботи з файлами і папками: створення папки, видалення, копіювання, вставка та інші.

Лістинг 4 знаходиться в головному процесі Electron — він відповідає за вікна програми і зв'язок з системою. Використовуються модулі ipcMain для отримання подій від інтерфейсу (рендер-процесу) і Menu для збірки і показу самого контекстного меню.

Лістинг 4 — *Контекстне меню “Main.ts”*

```
ipcMain.on(
    'show-context-menu',
    async (event, { filePaths, targetPath }) => {
      if (!mainWindow) return;
      const template: MenuItemConstructorOptions[] = [];
      template.push({
        label: 'New Folder',
        click: () => {
          mainWindow?.webContents.send('create-folder-
dialog');
        },
      });

      if (filePaths && filePaths.length > 0) {
        template.push(
          {
            label: 'Delete',
            click: async () => {
              for (const fp of filePaths) {
                await deleteFileOrFolder(fp);
```

```
                }
                mainWindow?.webContents.send('refresh-
files');
              },
            },
            {
              label: 'Copy',
              click: () => {
                clipboardPaths = filePaths;
                clipboardAction = 'copy';
              },
            },
          );
        }
        template.push({
          label: 'Paste',
          enabled: clipboardPaths !== null &&
clipboardPaths.length > 0,
          click: async () => {
            if (!clipboardPaths || !clipboardAction) return;
            let destinationDir: string = '';
            if (targetPath) {
              try {
                const stats = await
fs.promises.stat(targetPath);
                destinationDir = stats.isDirectory()
                  ? targetPath
                  : path.dirname(targetPath);
              } catch (err) {
                destinationDir = '';
              }
            }
            for (const cp of clipboardPaths) {
              let normalizedCp = cp;
              if (os.platform() === 'linux') {
                normalizedCp = cp.replace(/\\/g, '/');
              }
              const fileName = path.basename(normalizedCp);
              const newFilePath = await getUniqueFilePath(
                destinationDir,
                fileName,
              );
              try {
                const stats = await
fs.promises.stat(normalizedCp);
                if (stats.isDirectory()) {
                  await fs.promises.cp(normalizedCp,
newFilePath, {
                    recursive: true,
```

```
                });
              } else {
                await fs.promises.copyFile(normalizedCp,
newFilePath);
              }
            } catch (error) {
            }
          }
          mainWindow?.webContents.send('refresh-files');
        },
      });
      const menu = Menu.buildFromTemplate(template);
      menu.popup({
        window: mainWindow,
        x: undefined,
        y: undefined,
        positioningItem: 0,
      });
    },
  );
```

Лістинг 5 основного процесу (main process) в Electron-додатку та використовує модуль ipcMain для обробки запитів від рендер-процесу (renderer process). Давайте розберемо його покроково:

Лістинг 5 — *Відкриття файлів “maint.ts”*

```
ipcMain.handle('open-file', async (event, filePath) =>
{    const command =
      process.platform === 'win32'
        ? `start "" "${filePath}"`
        : process.platform === 'darwin'
          ? `open "${filePath}"`
          : `xdg-open "${filePath}"`; // Linux
    return new Promise<{ success: boolean; error?: string
}>((resolve) => {
      exec(command, (error: Error | null) => {
        if (error) {
          resolve({ success: false, error: error.message
});
        } else {
          resolve({ success: true });
        }
      });
    });
  });
```

Лістинг 6 налаштовує обробник для каналу 'create-folder' у головному процесі Electron за допомогою ipcMain.handle. Він дозволяє рендер-процесу (інтерфейсу користувача) надсилати запит на створення нової папки у файловій системі, передаючи два параметри: parentPath (шлях до батьківської папки) і folderName (назва нової папки). Обробник виконує створення папки і повертає результат — успішний чи з помилкою.

Лістинг 6 Створеня папки “main.ts”

```
ipcMain.handle('create-folder', async (event, params: {
parentPath: string; folderName: string }) => {
    try {
      if (!params || typeof params !== 'object') {
        return { success: false, error: 'Invalid parameters
format' };
      }
      const { parentPath, folderName } = params;
      if (!parentPath || !folderName) {
        return { success: false, error: 'Invalid path or
folder name' };
      }
      const folderPath = path.join(parentPath, folderName);
      await fs.promises.mkdir(folderPath, { recursive: true
});
      mainWindow?.webContents.send('refresh-files');
      return { success: true };
    } catch (error: unknown) {
      const errorMessage = error instanceof Error ?
error.message : String(error);
      return { success: false, error: errorMessage };
    }
  });
```

Лістинг 7 налаштовує обробник у головному процесі Electron для каналу 'get-parent'. Він отримує шлях currentPath від рендер-процесу (інтерфейсу користувача) і повертає шлях до батьківської папки. Наприклад:

Якщо currentPath — це /home/user на Linux, код поверне /home.

Якщо currentPath — це C:\Users\Me на Windows, код поверне C:\Users.

Обробник враховує особливості різних ОС (Windows, Linux, macOS), щоб правильно визначати батьківський шлях і повертати його у відповідному форматі (з \ для Windows або / для Linux/macOS).

Чому це потрібно для різних ОС?

Операційні системи мають різні способи представлення шляхів у файловій системі:

Windows: Використовує літери дисків (наприклад, C:) і зворотні слеші \ як роздільники.

Linux/macOS: Використовує єдиний корінь / і прямі слеші / як роздільники.

Лістинг 7 адаптується до цих відмінностей, щоб коректно обробляти шляхи на кожній платформі.

Лістинг 7 — *Шлях "main.ts"*

```
ipcMain.handle('get-parent', async (_, currentPath: string)
=> {
    const { platform } = process;
    if (platform === 'win32') {
      if (/^[A-Za-z]:\\$/.test(currentPath)) return
currentPath;
      let parentPath = path.dirname(currentPath);
      if (/^[A-Za-z]:$/.test(parentPath)) {
        parentPath += '\\';
      }
      return parentPath;
    }
    // Для Linux/macOS
    if (currentPath === '/') return currentPath;
    let parentPath = currentPath.replace(new
RegExp('/[^/]+/?$'), '');
    if (parentPath === '') {
      parentPath = '/';
    }
    return parentPath;
  });
```

Лістинг 8 основним React-компонентом файлового менеджера. Він включає:

Заголовок (Header): Логотип додатку та кнопки для перемикання режиму "книжки" і теми (світла/темна).

Область контенту (Content): Складається з бічної панелі (Sidebar) та однієї або двох файлових панелей (FilePanel), залежно від режиму.

Діалогові вікна: Для створення папок, роботи з тегами та вибору тегів для файлів.

Компонент об'єднує логіку роботи з файлами, вкладками, пошуком і тегами, передаючи пропси та обробники подій дочірнім компонентам.

Лістинг 8 — *Головний файл"FileExplorer.tsx"*

```
  const renderLeftPanel = () => (
    <FilePanel
      files={filterFilesBySearch(left.files,
leftSearchTerm)}
      tabs={left.tabs}
      currentTabId={left.currentTabId}
      currentTab={left.currentTab}
      isDarkTheme={isDarkTheme}
      selectedFiles={left.selectedFiles}
      hoveredFileIndex={left.hoveredFileIndex}
      selectedFileIndex={left.selectedFileIndex}
      dselectedFileIndex={left.dselectedFileIndex}
      anchorIndex={left.anchorIndex}
      isActivePanel={!isBookMode || activePage === 'left'}
      isLeftPanel
      pathSeparator={pathSeparator}
      onCreateFolderClick={() =>
setIsCreateFolderDialogOpen(true)}
      onAddTagClick={handleAddTagClick}
      onTabSelect={(tabId) => handleTabSelect(tabId, true)}
      onTabClose={left.removeTab}
      onAddTab={left.addTab}
      onBackClick={left.handleBackClick}
      onForwardClick={left.handleForwardClick}
      onParentClick={() => handleParentClick(true)}
      onRefreshClick={left.refreshFiles}
      onPathChange={left.setPathWithHistory}
      onEnterClick={() => fetchFiles(true)}
      onFileClick={(e, index, filePath) =>
handleFileClick(true, e, index, filePath)}
      onFileDoubleClick={(index) =>
handleFileDoubleClick(true, index)}
```

```
      onFolderOpen={(name, index) =>
handleFolderClick(name, true, index)}
      onFileOpen={handleFileOpen}
      onContextMenu={handleContextMenu}
      setHoveredFileIndex={left.setHoveredFileIndex}
      onSearch={setLeftSearchTerm}
      searchTerm={leftSearchTerm}
    />
  );

  const renderRightPanel = () => (
    <FilePanel
      files={filterFilesBySearch(right.files,
rightSearchTerm)}
      tabs={right.tabs}
      currentTabId={right.currentTabId}
      currentTab={right.currentTab}
      isDarkTheme={isDarkTheme}
      selectedFiles={right.selectedFiles}
      hoveredFileIndex={right.hoveredFileIndex}
      selectedFileIndex={right.selectedFileIndex}
      dselectedFileIndex={right.dselectedFileIndex}
      anchorIndex={right.anchorIndex}
      isActivePanel={activePage === 'right'}
      isLeftPanel={false}
      pathSeparator={pathSeparator}
      onCreateFolderClick={() =>
setIsCreateFolderDialogOpen(true)}
      onAddTagClick={handleAddTagClick}
      onTabSelect={(tabId) => handleTabSelect(tabId,
false)}
      onTabClose={right.removeTab}
      onAddTab={right.addTab}
      onBackClick={right.handleBackClick}
      onForwardClick={right.handleForwardClick}
      onParentClick={() => handleParentClick(false)}
      onRefreshClick={right.refreshFiles}
      onPathChange={right.setPathWithHistory}
      onEnterClick={() => fetchFiles(false)}
      onFileClick={(e, index, filePath) =>
handleFileClick(false, e, index, filePath)}
      onFileDoubleClick={(index) =>
handleFileDoubleClick(false, index)}
      onFolderOpen={(name, index) =>
handleFolderClick(name, false, index)}
      onFileOpen={handleFileOpen}
      onContextMenu={handleContextMenu}
      setHoveredFileIndex={right.setHoveredFileIndex}
      onSearch={setRightSearchTerm}
```

```
        searchTerm={rightSearchTerm}
      />
    );
    return (
      <div className={`Expl ${isDarkTheme ? 'dark-theme' :
'light-theme'}`}>
        <div className="Header">
          <div className="logo">
            <h1>Explorer</h1>
          </div>
          <div className="HeaderB">
            <button className="NOB" onClick={() =>
toggleBookMode(!isBookMode)}>
              {isBookMode ? (
                <Icon
                  icon={IconNames.WIDGET_FOOTER}
                  color={isDarkTheme ? '#d9dde0' : undefined}
                />
              ) : (
                <Icon
                  icon={IconNames.MANUAL}
                  color={isDarkTheme ? '#d9dde0' : undefined}
                />
              )}
            </button>
            <button className="NOB" onClick={toggleTheme}>
              {isDarkTheme ? (
                <Icon icon={IconNames.FLASH} color="#d9dde0"
/>
              ) : (
                <Icon icon={IconNames.MOON} />
              )}
            </button>
          </div>
        </div>
        <div className="Content">
          <Sidebar
            shortcuts={shortcuts}
            availableDisks={availableDisks}
            activePath={activePage === 'left'
              ? left.currentTab?.currentPath || ''
              : right.currentTab?.currentPath || ''}
            isDarkTheme={isDarkTheme}
            hoveredShortcutIndex={hoveredShortcutIndex}
            hoveredDiskIndex={hoveredDiskIndex}
            onShortcutClick={(path) =>
handleShortcutClick(path, activePage === 'left')}
            onDiskSelect={(disk) => handleDiskSelect(disk,
activePage === 'left')}
```

```
            setHoveredShortcutIndex={setHoveredShortcutIndex}
            setHoveredDiskIndex={setHoveredDiskIndex}
            getIconForShortcut={getIconForShortcut}
            onTagClick={handleTagClick}
          />
          {isBookMode ? (
            <div className="BookView">
              <div
                className={`Main1 ${activePage === 'left' ?
'active' : ''}`}
                onClick={() => setActivePage('left')}
              >
                {renderLeftPanel()}
              </div>
              <div
                className={`Main2 ${activePage === 'right' ?
'active' : ''}`}
                onClick={() => setActivePage('right')}
              >
                {renderRightPanel()}
              </div>
            </div>
          ) : (
            <div
classNamе="SingleView">{renderLeftPanel()}</div>
          )}
        </div>

        <CreateFolderDialog
          isOpen={isCreateFolderDialogOpen}
          folderName={newFolderName}
          onFolderNameChange={setNewFolderName}
          onCancel={() => setIsCreateFolderDialogOpen(false)}
          onCreate={() => handleCreateFolder(activePage ===
'left')}
        />

        <TagDialog
          isOpen={isTagDialogOpen}
          onClose={handleTagDialogClose}
          onSave={handleTagSave}
          tag={tagToEdit}
        />

        <TagSelectionDialog
          isOpen={isTagSelectionDialogOpen}
          onClose={handleTagSelectionDialogClose}
          onApplyTag={handleApplyTag}
          onCreateNewTag={handleCreateNewTag}
```

```
          selectedFilePaths={selectedFilePaths}
        />
      </div>
    );
};
```

Лістинг 9 частиною React-компонента файлового менеджера, побудованого на основі Electron. Він містить оголошення стану (state) за допомогою хука useState, а також використання кастомного хука useFilePanel для управління станом файлових панелей. Змінні, описані в коді, є глобальними для компонента і використовуються для управління інтерфейсом та логікою роботи з файлами.

Лістинг 9 — *Стамуси в React" FileExplorer.tsx "*

```
const [platform, setPlatform] = useState<string>('');
  const [pathSeparator, setPathSeparator] =
useState<string>('');
  const [isPlatformInitialized, setIsPlatformInitialized] =
useState<boolean>(false);

  // Theme state
  const { isDarkTheme, toggleTheme } = useTheme();

  // View mode state
  const [isBookMode, setIsBookMode] =
useState<boolean>(false);
  const [activePage, setActivePage] =
useState<ActivePage>('left');

  const [leftSearchTerm, setLeftSearchTerm] = useState('');
  const [rightSearchTerm, setRightSearchTerm] =
useState('');

  // File panels state
  const left = useFilePanel({ pathSeparator, platform });
  const right = useFilePanel({ pathSeparator, platform });

  // Sidebar state
  const [shortcuts, setShortcuts] =
useState<Shortcut[]>([]);
  const [availableDisks, setAvailableDisks] =
useState<string[]>([]);
```

```
  const [hoveredShortcutIndex, setHoveredShortcutIndex] =
useState<number | null>(null);
  const [hoveredDiskIndex, setHoveredDiskIndex] =
useState<number | null>(null);

  // Dialog state
  const [isCreateFolderDialogOpen,
setIsCreateFolderDialogOpen] = useState<boolean>(false);
  const [newFolderName, setNewFolderName] =
useState<string>('');

  // Tag dialog state
  const [isTagDialogOpen, setIsTagDialogOpen] =
useState<boolean>(false);
  const [tagToEdit, setTagToEdit] = useState<FileTag |
undefined>(undefined);
  const [selectedFilePaths, setSelectedFilePaths] =
useState<string[]>([]); //

  // Tag selection dialog state
  const [isTagSelectionDialogOpen,
setIsTagSelectionDialogOpen] = useState<boolean>(false);
```

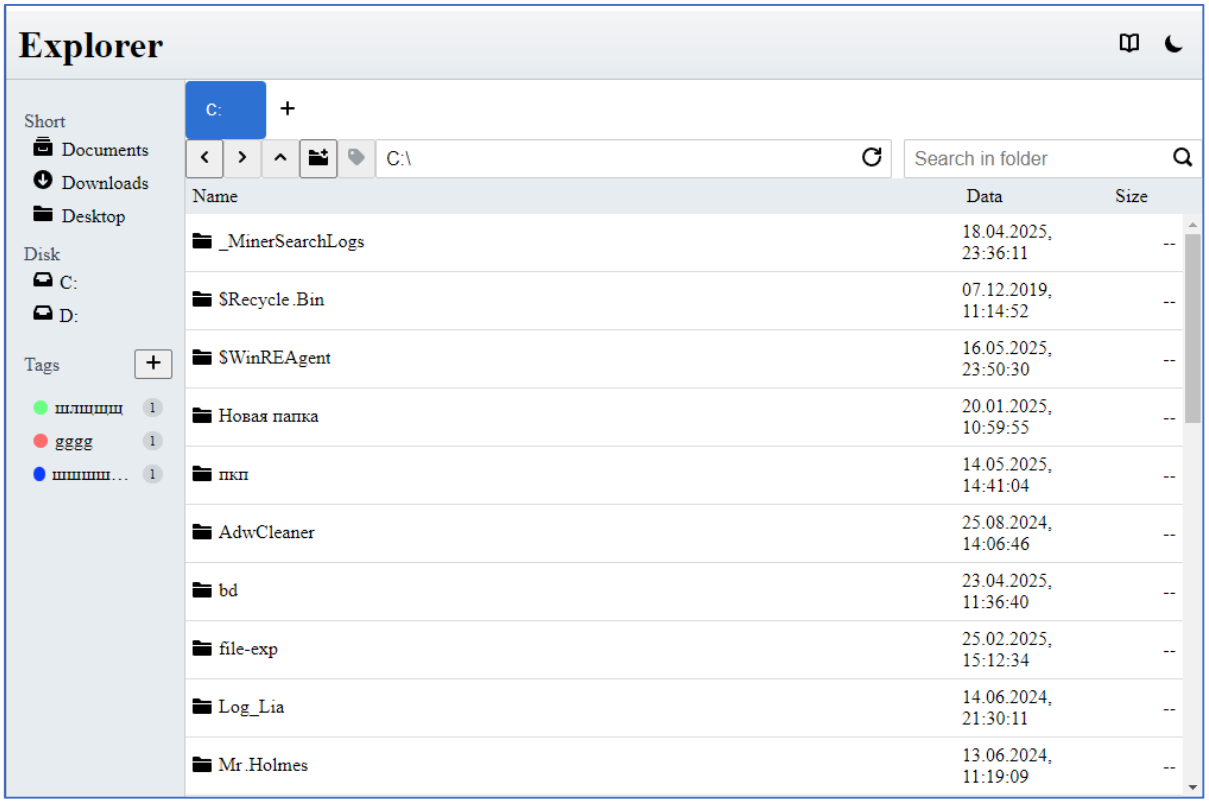


Рисунок 7 — *Explorer*

Лістинг 10 відповідає за рендеринг бокової панелі, яка містить:

Ярлики (Shortcuts): Список папок, які користувач позначив як "швидкий доступ" (наприклад, "Документи", "Завантаження").

Диски (Disks): Список доступних дисків у системі (наприклад, C:, D: на Windows або / на Linux/macOS).

Теги (Tags): Список тегів, які можна призначати файлам для їхньої категоризації, разом із можливістю додавання нових тегів через діалогове вікно.

Компонент використовує React для динамічного відображення елементів, реагує на взаємодію користувача (кліки, наведення миші) і підтримує темну/світлу тему. (див Рис 8)

Лістинг 10 — *Бокова панель "SideBar.tsx"*

```
return (
    <div className="left">
      <ul>
        <li>Short</li>
        {shortcuts.map((shortcut, index) => {
          const normalizedActivePath =
activePath.replace(/\\+$/, '');
          const normalizedShortcutPath =
shortcut.path.replace(/\\+$/, '');
          return (
            <li
              key={index}
              className={`quick-access-item ${
                normalizedActivePath ===
normalizedShortcutPath
                  ? 'active'
                  : ''
              } ${hoveredShortcutIndex === index ?
'hovered' : ''}`}
              onClick={() =>
onShortcutClick(shortcut.path)}
              onMouseEnter={() =>
setHoveredShortcutIndex(index)}
              onMouseLeave={() =>
setHoveredShortcutIndex(null)}
            >
              <Icon
                icon={getIconForShortcut(shortcut.name)}
```

```
                  style={{ marginRight: '8px' }}
                  color={isDarkTheme ? '#d9dde0' : undefined}
                />
                {shortcut.name}
              </li>
            );
          })}
          <li>Disk</li>
          {availableDisks.length > 0 ? (
            availableDisks.map((disk, index) => (
              <li
                key={disk}
                className={`disk-item ${hoveredDiskIndex ===
index ? 'hovered' : ''}`}
                onClick={() => onDiskSelect(disk)}
                onMouseEnter={() =>
setHoveredDiskIndex(index)}
                onMouseLeave={() =>
setHoveredDiskIndex(null)}
              >
                <Icon
                  icon={IconNames.INBOX}
                  color={isDarkTheme ? '#d9dde0' : undefined}
                  style={{ marginRight: '6px' }}
                />
                {disk}
              </li>
            ))
          ) : (
            <p>Диски не знайдені</p>
          )}
          <li className="section-header">
            <span>Tags</span>
            <Button
              minimal
              small
              icon={IconNames.PLUS}
              onClick={() => openTagDialog()}
              title="Добавить новый тег"
            />
          </li>
          {isTagsLoading ? (
            <li>Завантаження тегов...</li>
          ) : tags.length > 0 ? (
            tags.filter(tag => tag !== null).map((tag, index)
=> (
              <li
                key={tag?.id || `tag-${index}`}
```

```
                  className={`tag-item ${hoveredTagIndex ===
index ? 'hovered' : ''}`}
                  onClick={() => handleTagClick(tag)}
                  onMouseEnter={() =>
setHoveredTagIndex(index)}
                  onMouseLeave={() => setHoveredTagIndex(null)}
                >
                  <div
                    className="tag-color"
                    style={{ backgroundColor: tag?.color ||
'#cccccc' }}
                  />
                  <span className="tag-name">{tag?.name ||
'Безымянный тег'}</span>
                  <span className="tag-
count">{tag?.paths?.length || 0}</span>
                </li>
              ))
            ) : (
              <li className="empty-item">Теги не створені</li>
            )}
          </ul>

          <TagDialog
            isOpen={isTagDialogOpen}
            onClose={closeTagDialog}
            onSave={handleTagSave}
            tag={tagToEdit}
          />
        </div>
      );
};
```

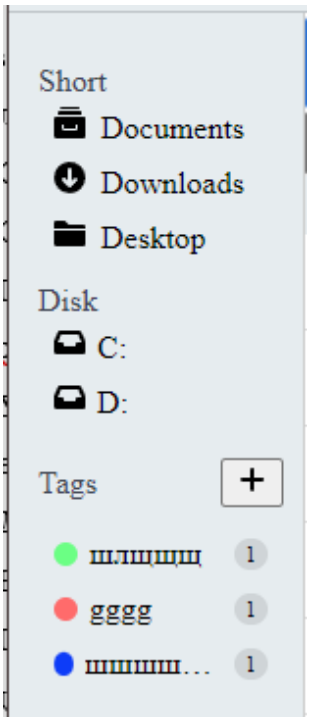


Рисунок 8 — *SideBar*

Лістинг 11 створює інтерфейс вкладок, схожий на систему вкладок у браузерах, де кожна вкладка може представляти окремий перегляд, документ або розділ у додатку. Вона відповідає за відображення заголовків вкладок та обробку взаємодії з користувачем (вибір вкладки, закриття, додавання), тоді як логіка перемикання вмісту та управління станом вкладок, реалізується у батьківській компоненті. (див Рис 9)

Лістинг 11 — *Вкладки“TabBar.tsx”*

```
return (
    <div className="Multopen">
        {tabs.map((tab) => {
          const tabStyle = tab.tagColor ? {
            borderLeft: `4px solid ${tab.tagColor}`,
            borderTopLeftRadius: '0',
            borderBottomLeftRadius: '0',
            paddingLeft: '6px'
          } : {};

          return (
            <div
              key={tab.id}
```

```
                  className={`tab ${tab.id === currentTabId ?
'active' : ''}`}
                  style={tabStyle}
                >
                  <button onClick={() => onTabSelect(tab.id)}>
                    {tab.name}
                  </button>
                  {tabs.length > 1 && (
                    <button
                      className="close-tab"
                      onClick={(e) => {
                        e.stopPropagation();
                        onTabClose(tab.id);
                      }}
                      type="button"
                    >
                      <Icon
                        icon={IconNames.CROSS}
                        color={isDarkTheme ? '#d9dde0' :
undefined}
                        size={12}
                      />
                    </button>
                  )}
                </div>
              );
            })}
            <button className="NOB" onClick={onAddTab}
type="button">
              <Icon
                icon={IconNames.PLUS}
                color={isDarkTheme ? '#d9dde0' : undefined}
              />
            </button>
        </div>
    );
};
```

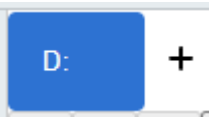


Рисунок 9 — *TabBar*

Лістінг 12 React-компоненти, яка реалізує панель дій для управління файлами та навігації. Ця компонента містить кнопки для навігації («назад»,

«вперед», «вгору»), створення папок, управління тегами, відображення та редагування шляху, а також пошуку файлів у поточній папці. (див Рис 10)

Лістінг 12 — *Панель дій за файлами “ PanelNavigation.tsx”*

```
return (
    <div className="User">
        <button
          type="button"
          onClick={onBackClick}
          disabled={isTagMode || !currentTab ||
currentTab.currentIndex === 0}
          title={isTagMode ? 'Навігація недоступна в режимі
тегів ' : 'Назад'}
        >
          <Icon
            icon={IconNames.CHEVRON_LEFT}
            color={isDarkTheme ? '#d9dde0' : undefined}
          />
        </button>
        <button
          type="button"
          onClick={onForwardClick}
          disabled={
            isTagMode ||
            !currentTab ||
            currentTab.currentIndex ===
currentTab.history.length - 1
          }
          title={isTagMode ? 'Навігація недоступна в режимі
тегів' : 'Вперед'}
        >
          <Icon
            icon={IconNames.CHEVRON_RIGHT}
            color={isDarkTheme ? '#d9dde0' : undefined}
          />
        </button>
        <button
          type="button"
          onClick={onParentClick}
          disabled={isTagMode || (!currentTab?.currentPath
|| currentTab.currentPath.endsWith(':\\'))}
          title={isTagMode ? 'Навігація недоступна в режимі
тегів: currentTab?.currentPath &&
currentTab.currentPath.endsWith(':\\') ? Немає батьківської
директорії ' : 'Вгору'}
        >
```

```
            <Icon
              icon={IconNames.CHEVRON_UP}
              color={isDarkTheme ? '#d9dde0' : undefined}
            />
          </button>
          <button type="button"
onClick={onCreateFolderClick}>
            <Icon
              icon={IconNames.FOLDER_NEW}
              color={isDarkTheme ? '#d9dde0' : undefined}
            />
          </button>
          <button
            type="button"
            onClick={onAddTagClick}
            disabled={!isAddTagEnabled}
            title={isTagMode
              ? isAddTagEnabled
                ? `Видалити тег "${currentTab?.name}" з
вибраних елементів `
                : 'Виберіть один або декілька файлів для
видалення тегу
              : isAddTagEnabled
                ? 'Додати тег до вибраних елементів'
                : 'Виберіть один або кілька файлів або папок'
            }
          >
            <Icon
              icon={isTagMode ? IconNames.DISABLE :
IconNames.TAG}
              color={
                isDarkTheme
                  ? isAddTagEnabled ? '#d9dde0' : '#5c7080'
                  : isAddTagEnabled ? undefined : '#99a0a5'
              }
            />
          </button>
        <div className="Path">
          <input
            placeholder="Path"
            className={`Input ${isTagMode ? 'tag-mode' :
''}`}
            value={isTagMode
              ? (path
                  ? path
                  : currentTab
                    ? `Тег: ${currentTab.name}`
                    : '')
              : (path || currentTab?.currentPath || '')}
```

```
          onChange={(e) => !isTagMode &&
onPathChange(e.target.value)}
          onKeyDown={(e) => !isTagMode && handleKeyDown(e)}
          readOnly={isTagMode}
          title={isTagMode ? 'У режимі тегів шлях
відображається, але редагувати його не можна' : ''}
        />
        <button className="NOBS" onClick={onRefreshClick}>
          <Icon
            icon={IconNames.REPEAT}
            color={isDarkTheme ? '#d9dde0' : undefined}
          />
        </button>
      </div>

      <div className="Search">
        <input
          placeholder="Search in folder"
          className="Input SearchInput"
          value={searchTerm || ''}
          onChange={(e) => onSearch &&
onSearch(e.target.value)}
          onKeyDown={(e) => e.key === 'Escape' && onSearch
&& onSearch('')}
        />
        <button className="NOBS" onClick={() => onSearch &&
onSearch('')}>
          {searchTerm ? (
            <Icon
              icon={IconNames.CROSS}
              color={isDarkTheme ? '#d9dde0' : undefined}
            />
          ) : (
            <Icon
              icon={IconNames.SEARCH}
              color={isDarkTheme ? '#d9dde0' : undefined}
            />
          )}
        </button>
      </div>
    </div>
  );
};
```

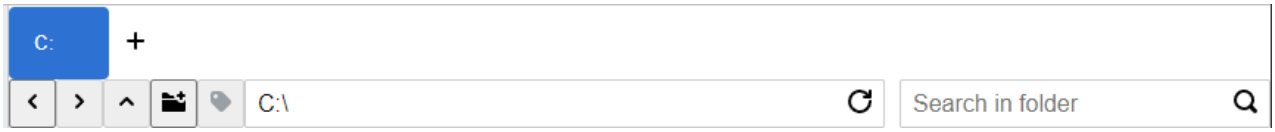


Рисунок 10 — *PanelNavigation*

Лістинг 13 React-компоненти, яка відображає один елемент (файл або папку) у списку файлів, як зазначено у вашому запиті. Цей код є частиною файлового менеджера і відповідає за виведення інформації про файли та папки, а також обробку взаємодії користувача з ними.

Цей компонен відображає один рядок у списку файлів або папок. Вона показує:

Ім'я файлу чи папки з відповідною іконкою.

Дату створення.

Розмір (для файлів) або позначку "--" (для папок).

Крім того, вона підтримує різні дії користувача, такі як вибір елемента, відкриття файлу чи папки, або виклик контекстного меню, та вигляд (див Рис 11)

Лістинг 13 — *Список файлів "FileItem.tsx"*

```
return (
    <p
      className={`file-item ${fileClass}`}
      onMouseEnter={() => onMouseEnter(index)}
      onMouseLeave={onMouseLeave}
      onClick={(e) => {
        if (!isActivePanel) return;
        onClick(e, index, filePath);
      }}
      onDoubleClick={() => onDoubleClick(file.isDirectory,
file.name, index)}
      onContextMenu={(e) => {
        e.stopPropagation();
        onContextMenu(e, filePath);
      }}
    >
      <span className="file-name">
        <Icon
```

```
            icon={file.isDirectory ? IconNames.FOLDER_CLOSE :
IconNames.DOCUMENT}
            style={{ marginRight: '6px' }}
            color={isDarkTheme ? '#d9dde0' : undefined}
          />
          <span className="file-name-base">{base}</span>
          <span className="file-extension">{ext}</span>
        </span>
        <span className="file-date">
          {file.creationDate
            ? new Date(file.creationDate).toLocaleString()
            : '-'}
        </span>
        <span className="file-size">
          {file.isDirectory ? '--' : formatBytes(file.size)}
        </span>
      </p>
    );
};
```

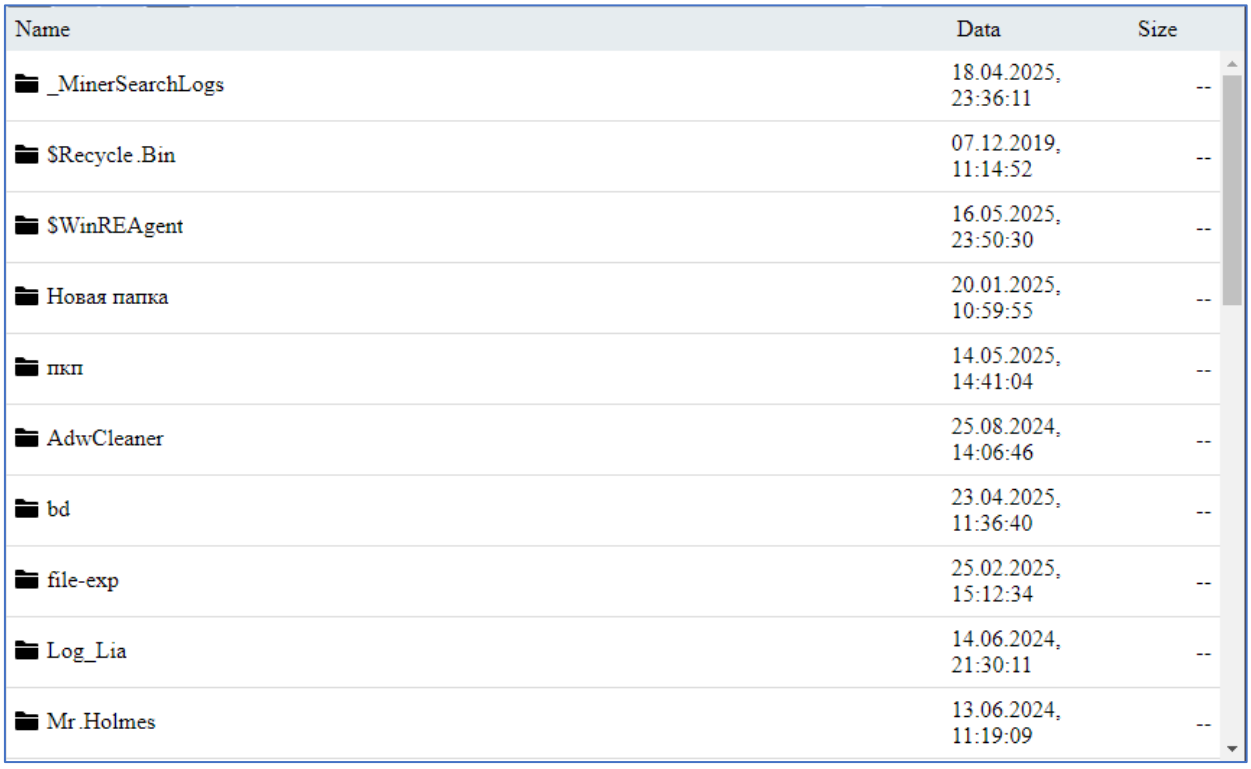


Рисунок 11 — *FileItem*

Лістинг 14 useTheme, який реалізує перемикання між світлою та темною темами в React-додатку, а також як він впливає на стиль інтерфейсу, як показано на рисунках (див Рис 12), (див Рис 11) та (див Рис 13)

Лістинг 14 — *Темна та світла тема”useTheme.ts”*

```
export const useTheme = () => {
  const [isDarkTheme, setIsDarkTheme] = useState(false);

  useEffect(() => {
    const savedTheme = localStorage.getItem('theme');
    if (savedTheme === 'dark') {
      setIsDarkTheme(true);
    }
  }, []);

  const toggleTheme = useCallback(() => {
    setIsDarkTheme(prev => {
      const newTheme = !prev;
      localStorage.setItem('theme', newTheme ? 'dark' :
'light');
      return newTheme;
    });
  }, []);

  return { isDarkTheme, toggleTheme };
};
```



Рисунок 12 — *Темна тема та книжка*

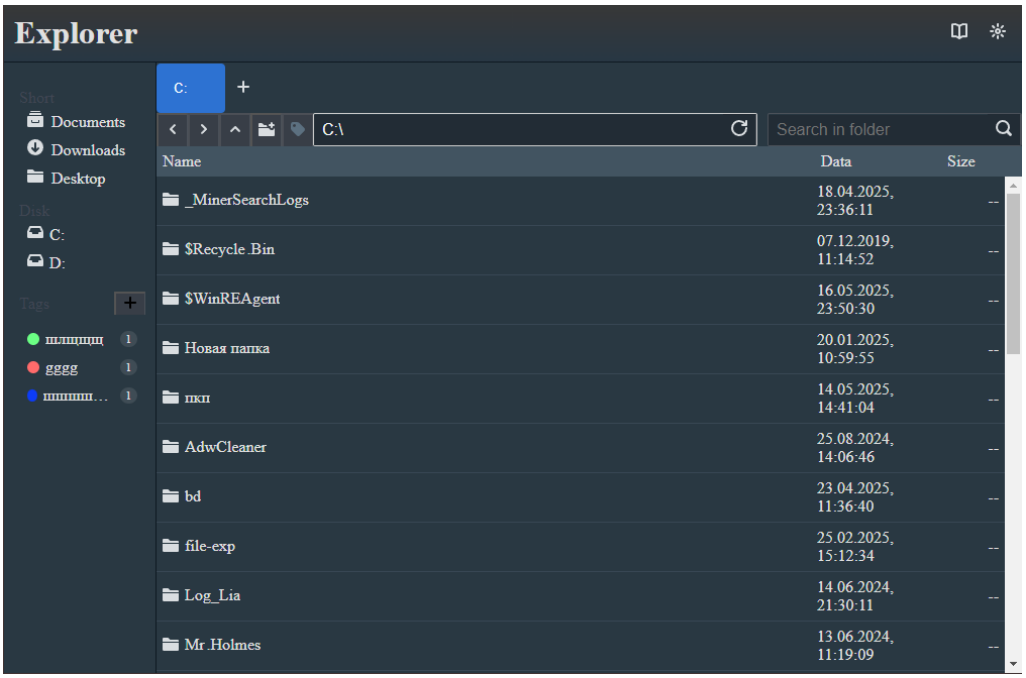


Рисунок 13 — *Вигляд темної теми*

Лістінг 15 Розглянемо функцію loadTagsFromFile, яка завантажує теги з JSON-файлу, парсить їх у форматі UTF-8 і повертає як масив об’єктів типу FileTag. Ця функція має вбудовану обробку помилок і логування для відстеження процесу. Давайте розберемо, як вона працює, і як теги завантажуються та парсяться з JSON-файлу. (див Рис 14)

Лістінг 15 — *Завантаження тегів “taghelper.ts”*

```
const loadTagsFromFile = (): FileTag[] => {
  try {
    const filePath = getTagsFilePath();
    console.log('Loading tags from:', filePath);
    if (!fs.existsSync(filePath)) {
      console.log('Tags file does not exist, creating empty
file');
      fs.writeFileSync(filePath, JSON.stringify([], null,
2), 'utf-8');
      return [];
    }

    const tagsData = fs.readFileSync(filePath, 'utf-8');
    console.log('Raw tags data:', tagsData);
    const parsedData = JSON.parse(tagsData);
```

```
    if (!Array.isArray(parsedData)) {
      console.error('Tags data is not an array, resetting
to empty array');
      fs.writeFileSync(filePath, JSON.stringify([], null,
2), 'utf-8');
      return [];
    }

    console.log('Successfully loaded tags:',
parsedData.length);
    return parsedData;
  } catch (error) {
    console.error('Error loading tags:', error);
    try {
      const filePath = getTagsFilePath();
      console.log('Recreating tags file due to error');
      fs.writeFileSync(filePath, JSON.stringify([], null,
2), 'utf-8');
    } catch (writeError) {
      console.error('Error recreating tags file:',
writeError);
    }

    return [];
  }
};
```

```
Loading tags from: C:\Users\HolyFreedom\AppData\R
Raw tags data: [
  {
    "id": "9bf6e441-41ef-4510-9d1b-95dbd48fd905",
    "name": "шлщщщ",
    "color": "#6bff84",
    "targetType": "both",
    "paths": [
      "D:\\cwi"
    ]
  },
```

Рисунок 14 — *Лог завантажженя тегов*

Лістінг 16 initTagsCache, яка відіграє важливу роль у роботі системи тегів, забезпечуючи її функціонування в реальному часі.

Лістинг 16 — *Кєшування “taghelper.ts”*

```
export const initTagsCache = (): void => {
  if (!isTagsCacheInitialized) {
    tagsCache = loadTagsFromFile();
    isTagsCacheInitialized = true;
    console.log('Tags cache initialized with',
tagsCache.length, 'tags');
  }
};
```

Система тегів

Зберігання тегів. Теги зберігаються в JSON-файлі (tags.json), що робить їх зручними для додавання, редагування та видалення.

Зв'язок з файлами. Один тег може бути прикріплений до декількох файлів за їх шляхами, а файл може мати кілька тегів. Це працює через функції applyTagToFile і removeTagFromFiles, які викликаються через IPC в Electron.

Режим тегів. Користувач може увімкнути цей режим, натиснувши на тег у бічній панелі (handleTagClick). Увімкнувши його, видно тільки ті файли, до яких прикріплений вибраний тег. Відбір відбувається в loadTagFiles: шляхи завантажуються, перевіряється наявність файлів, потім вони сортуються (спочатку папки, потім за алфавітом).

Робота з тегами. Є вікна для створення нового тегу (TagDialog) і вибору вже існуючого (TagSelectionDialog). Через них можна додавати теги до вибраних файлів або прибирати їх.

Зручне групування. Теги допомагають збирати файли за темами (наприклад, «Робота», «Фото») і швидко знаходити потрібне, не спираючись на структуру папок.

Фільтрація. У режимі тегів видно тільки потрібні файли, так що можна зосередитися на одному.

Простота дій. Теги можна додавати і прибирати відразу у декількох файлів, все зрозуміло і швидко.

Система історії

Хід по папках. У кожної панелі (ліва і права) є своя історія — вона зберігається в currentTab.history. Функція setPathWithHistory додає шлях в історію і оновлює поточний індекс, щоб можна було переміщатися назад і вперед.

Запам'ятовування. Коли користувач змінює вкладки (handleTabSelect), зберігається шлях та інші дані (наприклад, виділені файли). Якщо був увімкнений режим тегів, історія допомагає повернутися до звичайної папки після його вимкнення.

Навігація. Кнопка «вгору» (handleParentClick) і подвійне клацання по папці (handleFileDoubleClick) змінюють шлях з урахуванням історії. Після виходу з режиму тегів можна повернутися туди, де робота була перервана.

Швидке переміщення. Можна повернутися в попередні папки або піти далі, не втрачаючи поточного положення.

Робота в декількох місцях. Окрема історія для кожної вкладки дозволяє тримати відкритими різні папки і спокійно між ними перемикатися.

Стійкість. При переході між режимами зберігається контекст, що робить роботу більш плавною.

Система книжкового режиму

Дві панелі. При включенні книжкового режиму (toggleBookMode) з'являються відразу дві панелі — ліва і права. Змінна isBookMode керує їх відображенням, а activePage вказує, в якій панелі відбувається дія.

Запуск. Якщо права панель при включенні книжкового режиму порожня, вона отримує нову вкладку автоматично, як показано на рис.15 (через useEffect, що залежить від isBookMode).

Операції між панелями

Файли можна копіювати або переносити з однієї панелі в іншу через контекстне меню (`handleContextMenu`). Воно відправляє команди в основний процес Electron, де виконуються необхідні дії.

Простота дій. З двома панелями легко переміщати або копіювати файли — не потрібно переходити між вікнами або вкладками.

Порівняння. Одночасно видно дві папки, що зручно для звірки та перенесення даних.

Швидка робота. Коли перед очима відразу дві директорії, завдання на зразок копіювання або перенесення файлів виконуються швидше.

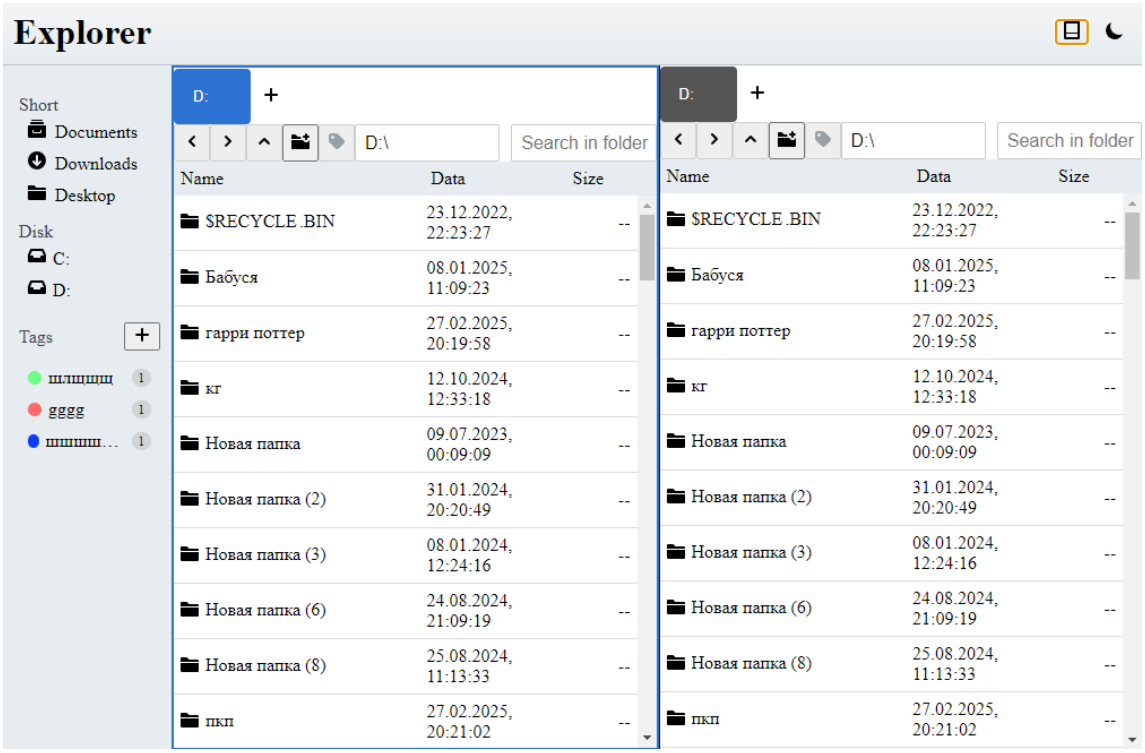


Рисунок 15 — *Вигляд у режимі книжки*

е. Тестування

Перевірка на інших ОС

Тестування програмного забезпечення на різних операційних системах (ОС) є ключовим етапом розробки Таблиця 2, щоб переконатися в коректній роботі програми в різних середовищах. У нашому випадку тестування проводилося на Linux за допомогою віртуалізації через VMware a macOS та Windows 11 з архітектурою ARM, на інших системах. Для створення програми використовувалася команда npm run package, яка збирає проект без налагоджувальної інформації.

Для тестування на різних операційних системах ви використали VMware — потужний інструмент віртуалізації, який дозволяє запускати кілька ОС на одному фізичному комп’ютері. VMware на яку встановлено Linux. На базі VMware створили віртуальну машину для тестування програми на інших ОС, зокрема macOS та Windows 11 з архітектурою ARM на інших системах. (див Рис. 16, Рис. 17, Рис. 18).

Для створення програми ви використовували команду npm run package. Ця команда є частиною екосистеми Node.js і запускає скрипт, визначений у файлі package.json вашого проекту.

Таблиця 2 — *Тестування*

Модуль / Функціонал	**Сценарії тестування**	**Очікуваний результат**
Панель інструментів	1. Перевірка роботи кнопок навігації («назад», «вперед», «вгору»). 2. Створення нової директорії. 3. Перевірка коректності відображення поточного шляху та його нормалізації для різних ОС. 4. Функціонування пошуку. 5. Створення, переключення та видалення вкладок.	Усі операції виконуються коректно, шлях та вміст панелі оновлюються відповідно до дій користувача
Контекстне меню	1. Копіювання, вставка та видалення одного та групи файлів. 2. Виконання операцій між різними дисками та директоріями.	Файли коректно копіюються, переміщуються та видаляються. Система обробляє операції між різними томами.
Двопанельний режим	1. Коректне переключення режиму (ввімкнення/вимкнення). 2. Перевірка незалежності історії навігації для лівої та правої панелей. 3. Збереження стану та історії для кожної вкладки в обох панелях.	Режим активується/деактивується коректно. Кожна панель зберігає свою унікальну історію та контекст.
Список файлів	1. Коректне відображення назв, іконок, розміру та дати модифікації файлів. 2. Обробка виділення одного та кількох файлів (з Ctrl/Shift). 3. Перевірка доступу до файлів (відкриття подвійним кліком).	Всі елементи списку відображаються правильно. Механізм виділення працює очікувано. Файли відкриваються асоційованими програмами.
Система тегів	1. Створення нового тегу через діалогове вікно. 2. Застосування існуючих тегів до файлів. 3. Активація режиму тегів та коректне відображення відфільтрованого списку файлів. 4. Видалення тегу з файлу..	Теги успішно створюються, застосовуються та видаляються. Фільтрація за тегами працює коректно.
Теми інтерфейсу	1. Переключення між світлою та темною темами. 2. Коректне застосування стилів до всіх елементів інтерфейсу.	Інтерфейс коректно змінює колірну схему без візуальних артефактів.

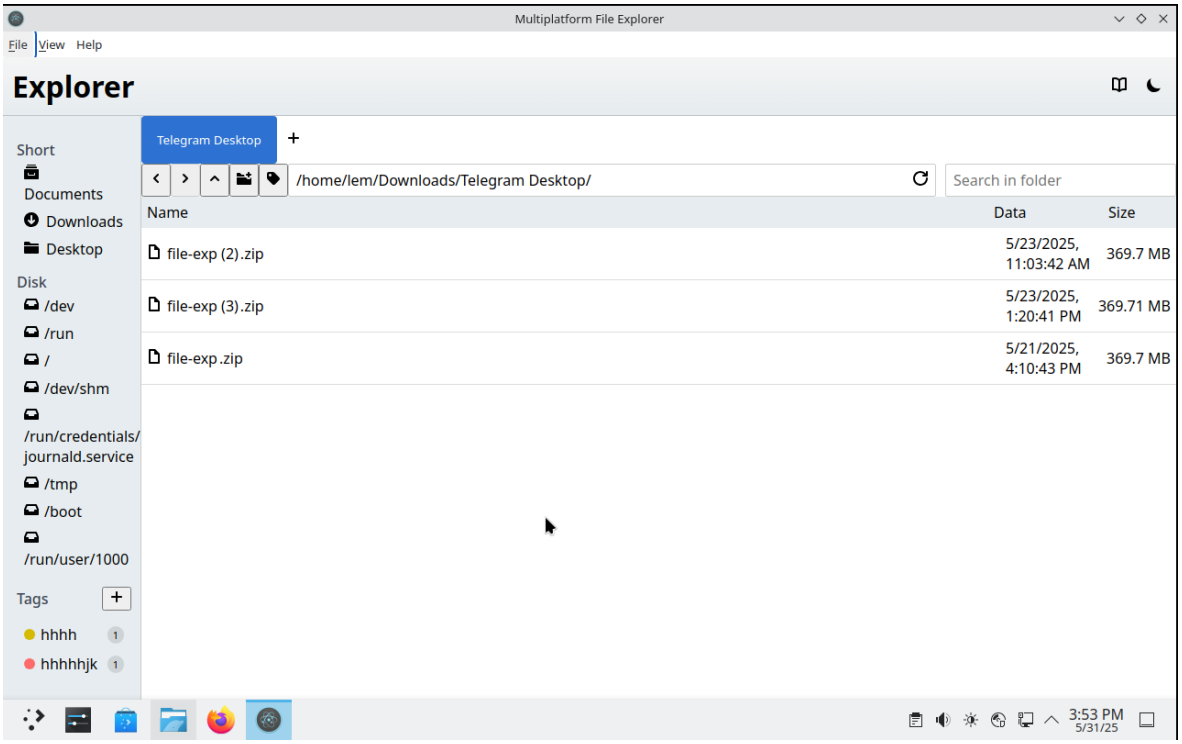


Рисунок 16 — *Linux*

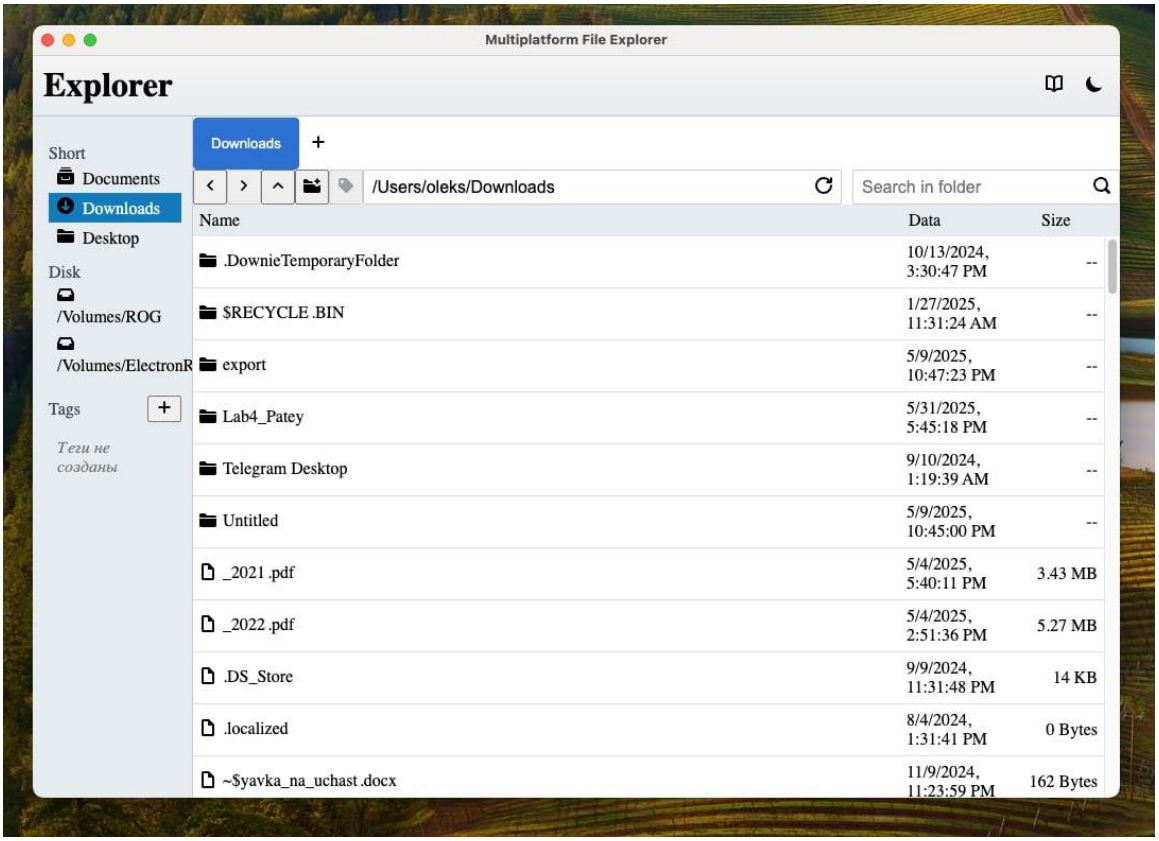


Рисунок 17 — *MacOS ARM*

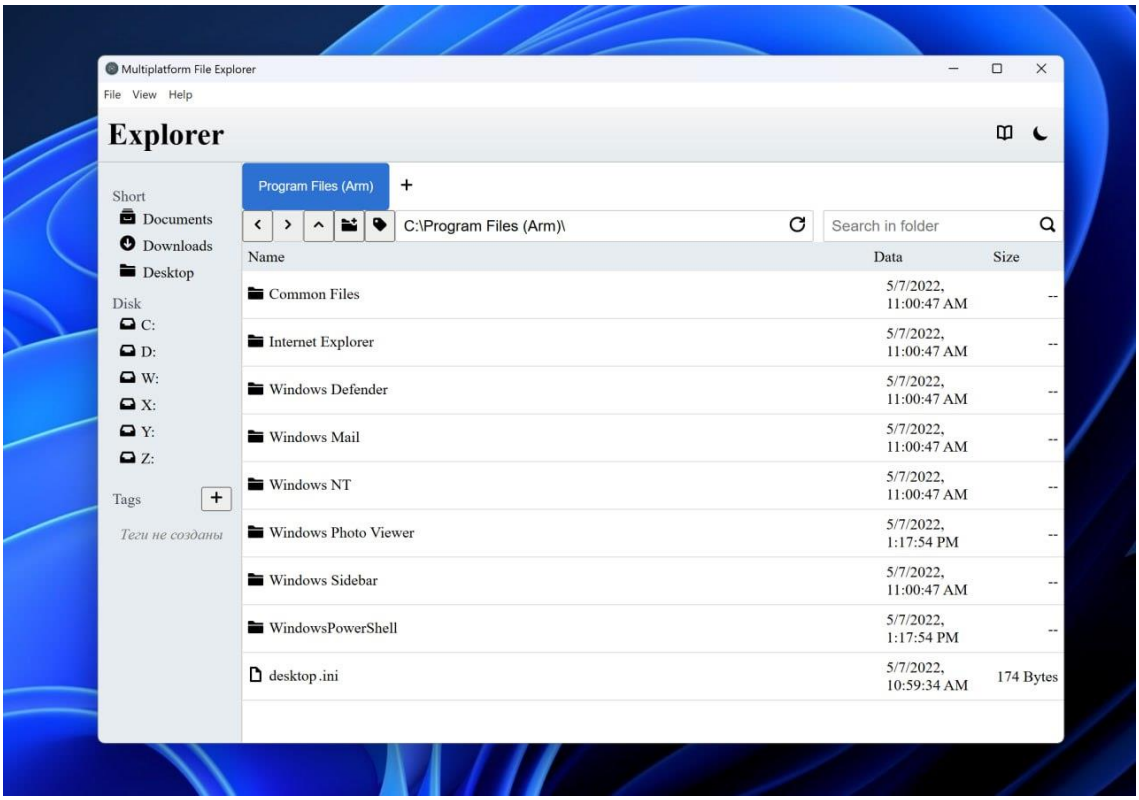


Рисунок 18 — *Windwos 11 ARM*

f. Висновки до розділу 3

Проведений аналіз предметної області та вимог показав, що для розробки мультиплатформного файлового експлорера оптимальним вибором є використання сучасних технологій, таких як Electron і React. Electron забезпечує кросплатформність і простий доступ до файлової системи, що є ключовим для роботи з файлами та тегами. React, у свою чергу, дозволяє реалізувати зручний, інтуїтивно зрозумілий і продуктивний інтерфейс користувача, що відповідає основним вимогам до UI.

Для забезпечення коректної роботи з файловою системою було розглянуто практичні аспекти реалізації, включаючи операції з файлами та тегами, а також проведено тестування для гарантії стабільності й надійності додатка. Використання таких бібліотек, як blueprintjs, полегшує розробку складного інтерфейсу, а node-disk-info надає необхідну інформацію про диски, що підсилює функціональність експлорера. Зберігання тегів у форматі JSON додає гнучкості та простоти в управлінні даними

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено мультиплатформний file explorer з використанням технологій Electron і React, що дозволяє користувачам ефективно працювати з файловою системою на операційних системах Windows, macOS та Linux. Додаток успішно реалізує ключові функції, такі як перегляд файлів і тек, тегування файлів для зручної організації, а також швидкий доступ до основних директорій і дисків. Використання Electron забезпечило кросплатформність і доступ до файлової системи, а React дозволив створити зручний та інтуїтивний інтерфейс користувача.

Основні досягнення роботи:

- *Кросплатформність.* Додаток успішно працює на різних операційних системах з єдиним кодом, що спрощує підтримку та розгортання.
- *Тегування файлів.* Реалізовано систему тегів за допомогою JSON, що дозволяє користувачам легко організовувати та фільтрувати файли.
- *Інтерфейс.* Використання бібліотеки blueprintjs забезпечило сучасний і зручний дизайн, а адаптивність інтерфейсу гарантує коректне відображення на різних пристроях.
- *Продуктивність.* Оптимізація через асинхронні запити та віртуалізацію списків дозволила досягти швидкої роботи навіть з великими директоріями.

Напрямки для подальшого вдосконалення:

- *Розширення функціональності.* Додати підтримку хмарних сховищ (наприклад, Google Drive, Dropbox) для інтеграції з віддаленими файлами.
- *Покращення тегування.* Реалізувати більш складну систему тегів з ієрархією або кольоровими мітками, як у macOS Finder.
- *Безпека.* Додати шифрування для захисту даних користувача, особливо при роботі з чутливими файлами.

Загалом, розроблений file explorer є функціональним і зручним інструментом, який може бути корисним для користувачів, що працюють на різних платформах. Проєкт продемонстрував успішне застосування сучасних технологій для створення настільних додатків і відкрив шлях для подальших досліджень у галузі кросплатформного програмування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. CSS // MDN Web Docs.URL: https://developer.mozilla.org/ru/docs/Web/CSS(дата звернення: 05.06.25).
2. Electron Documentation URL: https://www.electronjs.org/ru/docs/latest/README(дата звернення: 05.06.25).
3. Electron React Boilerplate. URL: https://electron-react-boilerplate.js.org/docs/installation(дата звернення: 05.06.25).
4. Getting Started // Reactjs.org. URL: https://ru.legacy.reactjs.org/docs/getting-started.html(дата звернення: 05.06.25).
5. HTML CSS // W3Schools. URL: https://www.w3schools.com/html/html_css.asp(дата звернення: 05.06.25).
6. Node.js Documentation URL: https://nodejs.org/docs/latest/api/(дата звернення: 05.06.25).
7. The TypeScript Handbook URL: https://www.typescriptlang.org/docs/handbook/intro.html(дата звернення: 05.06.25).
8. Вступ до HTML5 // W3Schools українською.URL: https://w3schoolsua.github.io/html/html5_intro.html#gsc.tab=0(дата звернення: 05.06.25).
9. Web-програмування - Кафедра програмного забезпечення автоматизованих систем // Запорізький національний університет.URL: https://sites.znu.edu.ua/webprog/ (дата звернення: 05.06.25).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Волович Андрій Володимирович, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення освітньої програми “Програмне забезпечення систем”, адреса електронної пошти se121-33@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Мультиплатформенний оглядач файлів з розширеним функціоналом**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 02.06.2025 _____ А. В. Волович
(підпис) (прізвище та ініціали) (студент)

Дата 02.06.2025 _____ О.О. Жуков
(підпис) (прізвище та ініціали) (керівник)