МІНІСТЕРСТВО ВНУТРІШНІХ СПРАВ УКРАЇНИ

ЛЬВІВСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ВНУТРІШНІХ СПРАВ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ УПРАВЛІННЯ,

ПСИХОЛОГІЇ ТА БЕЗПЕКИ

Кафедра інформаційних технологій

«РОЗРОБЛЕННЯ РОЗШИРЕННЯ В БРАУЗЕРІ ДЛЯ ПРОТИДІЇ З ФЕЙКОВИМИ НОВИНАМИ»

кваліфікаційна робота
здобувача вищої освіти
4 курсу денної форми навчання
Софії КАЛИН

Науковий керівник:
доцент, кандидат технічних наук
Ярко КУЛЕШНИК

Рецензент:
доцент, кандидат технічних наук
Ірина БОРЕЦЬКА

Кваліфікаційна робота допущена до захисту

«___» _____ 2026 р., протокол № ____

Завідувач кафедри інформаційних технологій

_____ **Олег ЗАЧЕК**
(підпис)

Львів – 2026

АНОТАЦІЯ

КАЛИН С. Розроблення розширення в браузері для протидії з фейковими новинами. – Рукопис.

Дослідження на здобуття освітнього ступеня ″бакалавр″ за спеціальністю 126 ″Інформаційні системи та технології″. – Львівський державний університет внутрішніх справ, МВС України, Львів, 2026.

Дипломна робота присвячена розробці актуальної теми сьогодення – створенню програмного забезпечення для боротьби з фейковими новинами та протидії дезінформації. У межах дослідження обґрунтовано актуальність вибраного проєкту та реалізовано інструментарій для верифікування медіаресурсів. Для розроблення архітектури системи використано фреймворк Svelte та середовище Node.js, що забезпечує високу реактивність інтерфейсу.

Для створення та управління базою даних застосовано Firebase Realtime Database, що дозволяє синхронізувати інформацію в реальному часі. Функціонал програми передбачає модулі реєстрації та автентифікування користувачів, а також спеціалізований інтерфейс для залишення відгуків про джерела інформації. Візуальне рішення інтерфейсу розроблено в мінімалістичному стилі.

Ключові слова: Svelte, програмне забезпечення, фреймворк, Node.js, Firebase, інтерфейс.

ABSTRACT

KALYN S. Development of a browser extension to combat fake news. – Manuscript.

Research for the degree of "bachelor" in specialty 126 "Information systems and technologies". – Lviv State University of Internal Affairs, Ministry of Internal Affairs of Ukraine, Lviv, 2026.

The diploma thesis is devoted to the development of a highly relevant contemporary topic – the creation of software to combat fake news and counteract disinformation. Within the scope of the study, the relevance of the chosen project is substantiated, and a toolkit for the verification of media resources is implemented. The Svelte framework and Node.js runtime environment were used to develop the system architecture, ensuring high interface reactivity.

Firebase Realtime Database was employed for database creation and management, allowing for real-time information synchronization. The software functionality includes user registration and authentication modules, as well as a specialized interface for submitting reviews regarding information sources. The visual design of the interface is executed in a minimalist style.

Keywords: Svelte, software, framework, Node.js, Firebase, interface.

ЗМІСТ

ВСТУП

Стрімка еволюція цифрових комунікаційних технологій призвела до суттєвого зростання обсягів інформації, що циркулює у мережі Інтернет, та радикально змінила механізми її продукування, поширення й споживання. В умовах децентралізованості інформаційного простору та зниження ролі традиційних медіа-інститутів зростає ймовірність появи та масштабування недостовірного або свідомо маніпулятивного контенту. Феномен фейкових новин виявився не лише соціально-комунікаційною проблемою, а й об’єктом дослідження в межах інформатики, когнітивних наук, соціології та кібербезпеки, оскільки його існування безпосередньо пов’язане з алгоритмічними процесами поширення даних, особливостями поведінки користувачів і властивостями цифрових платформ. З огляду на комплексний характер проблеми постає необхідність застосування технологічних підходів для її вирішення, зокрема інструментів аналізу, систематизації та верифікування інформації. Розроблення програмного забезпечення, здатного підтримувати функціонал виявлення, класифікації та оцінювання інформаційних джерел, становить важливий напрям досліджень у контексті підвищення інформаційної безпеки та формування надійного комунікаційного середовища. У межах даної роботи розглядаються теоретичні засади й технологічні рішення, необхідні для створення вебзастосунку, спрямованого на мінімізацію впливу фейкових новин у цифровому просторі.

Актуальність роботи полягає не тільки у виконанні аналізу предметної області, а також створенню програмного забезпечення для протидії фейковим новинам та дезінформації в умовах війни. Тема кваліфікаційої роботи відповідає сучасним трендам у розробленні програмного забезпечення протидії фейковим новинам та дезінформації на основі фреймворку Svelte, середовища Node.js, що забезпечує високу реактивність інтерфейсу і має безпосереднє практичне значення.

Метою дослідження є розроблення розширення для веббраузера, яке забезпечуватиме користувача швидким і зручним доступом до базових відомостей про джерело новинного матеріалу.

Об’єктом дослідження є застосування фреймворку «Svelte» у процесі створення браузерного розширення, призначеного для протидії поширенню недостовірної інформації.

Предметом дослідження є технологічні підходи та програмні рішення, що лежать в основі реалізування браузерного розширення з використанням фреймворку «Svelte».

Робота полягає у створенні зручного та ненавантаженого рекламою інтерфейсу, який дозволить користувачеві оперативно оцінювати надійність інформації, не витрачаючи час на самостійний аналіз потенційно фейкових матеріалів.

Практична цінність полягає у розробленні програмного забезпечення, спрямованого на виявлення та обмеження поширення фейкових новин, із використанням фреймворку Svelte для клієнтської частини та середовища NodeJS для серверної логіки. Система бази даних реалізується на основі Firebase Realtime Database.

Функціонал має охоплювати реєстрацію та аутентифікацію користувачів, а також надання можливості залишати відгуки щодо джерел інформації з подальшою модерацією адміністратором. У процесі розроблення застосовуються інструменти Rollup, Babel, Yarn і Tailwind.

Структура дипломної роботи охоплює вступ, три розділи, підсумкові висновки та перелік використаних джерел.

У першому розділі здійснено дослідження предметної області та аналіз існуючих програмних засобів та сервісів-аналогів. Другий розділ присвячено обгрунтуванню вибору програмного забезпечення. У третьому розділі розглянуто реалізування програмної частини, налаштування технологій та розроблення логіки розширень.

Загальний обсяг роботи становить 61 сторінку. У роботі подано 35 рисунків. Для підготовки дослідження використано 11 джерел. Наявний 1 додаток.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд проблемної області

Стрімке зростання обсягів цифрового контенту та доступність інструментів для його поширення зумовили суттєве ускладнення інформаційного середовища. У цьому контексті проблема розповсюдження недостовірних або свідомо маніпулятивних повідомлень набула міждисциплінарного характеру, охоплюючи сфери інформатики, соціальних комунікацій, медіаграмотності та кібербезпеки. Фейкові новини формуються та поширюються з використанням алгоритмів персоналізації, вірусних механізмів та психологічних ефектів, що сприяють швидкому проникненню неправдивої інформації у масову свідомість.

Традиційні методи перевірки фактів виявляються недостатніми в умовах високої динаміки інформаційних потоків, що потребує розроблення автоматизованих інструментів підтримки користувача під час оцінювання достовірності даних.

Одним із перспективних напрямів вирішення цієї проблеми є створення програмних засобів, інтегрованих у браузерне середовище, що дозволяють отримувати додаткові метадані щодо джерел новин безпосередньо під час споживання інформації. Браузерні розширення є зручним форматом, оскільки вони функціонують у межах типової поведінкової моделі користувача та не потребують переходу до сторонніх сервісів [1].

Використання сучасних фреймворків, зокрема «Svelte», забезпечує можливість створення високопродуктивних, легковагових та інтуїтивно зрозумілих інтерфейсів, адаптованих до вимог сучасних вебтехнологій.

Таким чином, проблемна область охоплює комплекс питань, пов’язаних із аналізом інформаційних загроз, оцінюванням надійності новинних джерел, інтеграцією таких механізмів у браузерний простір та вибором технологічних

рішень, оптимальних для реалізування швидкодіючого та коректного інструменту боротьби з недостовірним контентом.

1.2. Аналіз існуючих програмних засобів та сервісів-аналогів

Сучасний арсенал інструментів протидії дезінформації представлений переважно спеціалізованими сервісами в месенджерах та вбудованими алгоритмами соціальних мереж, кожен з яких базується на власній методології верифікування контенту. Одним із показових прикладів реактивного підходу є чат-бот «Перевірка», функціонування якого ґрунтується на поєднанні експертного аналізу та алгоритмів штучного інтелекту. Даний інструмент орієнтований на оперативне виявлення російської пропаганди та маніпулятивних повідомлень у соціальних медіа шляхом прямої взаємодії з користувачем, який має самостійно ініціювати процес перевірки, надсилаючи підозріле посилання. Попри високу точність висновків, зумовлену залученням фахівців-фактчекерів, суттєвим обмеженням подібних рішень є необхідність активної дії з боку споживача інформації та певна часова затримка в отриманні результату, що не завжди відповідає вимогам до швидкості споживання новинного контенту в сучасному цифровому середовищі [2].

Паралельно з автономними ботами розвиваються внутрішні системи верифікування глобальних платформ, таких як Meta та X (Twitter), які інтегрують механізми перевірки фактів безпосередньо в інтерфейс користувача. У межах екосистеми Facebook реалізовано стратегію партнерства з незалежними організаціями, зокрема StopFake, що дозволяє маркувати недостовірні дописи, знижувати їхній пріоритет у стрічці та обмежувати можливості монетизації для джерел, які систематично поширюють фейки. Аналогічно, мережа X впроваджує систему контекстуальних позначок та репутаційних санкцій, аж до безстрокового блокування облікових записів за рецидиви дезінформації. Проте ефективність таких заходів обмежена виключно межами конкретної платформи, що залишає значний сегмент

відкритого вебпростору, включно з новинними сайтами та блогами, поза межами автоматизованого контролю, створюючи ризики для користувача під час переходу за зовнішніми посиланнями.

Узагальнюючи стан ринку програмного забезпечення у сфері інформаційної безпеки, слід констатувати відсутність універсальних, масових рішень для браузерного середовища, які б функціонували за аналогією до популярних блокувальників рекламного контенту. Більшість існуючих програм є локальними за своєю природою та зосереджені в сегменті месенджерів або закритих соціальних екосистем, тоді як проблема верифікування автономних вебсайтів залишається технологічно нерозв'язаною на рівні стандартного ПЗ браузерів. Такий технологічний розрив обґрунтовує доцільність розроблення інтегрованого інструменту на базі сучасних фреймворків, як-от Svelte, що дозволить автоматизувати процес оцінювання надійності джерел безпосередньо в процесі серфінгу, забезпечуючи безперервний моніторинг інформаційного середовища без необхідності звернення до сторонніх сервісів [3].

ВИСНОВКИ ДО РОЗДІЛУ

У результаті комплексного аналізу проблемної області та існуючих засобів протидії дезінформації встановлено, що стрімке ускладнення сучасного інформаційного середовища та розповсюдження маніпулятивного контенту набули міждисциплінарного характеру, створюючи суттєві загрози кібербезпеці. Дослідження діючих програмних аналогів, зокрема чат-ботів та внутрішніх алгоритмів соціальних мереж Meta та X, виявило їхню обмежену ефективність через реактивну модель взаємодії, часову затримку в отриманні результатів та локалізацію виключно в межах закритих екосистем. Існуючий технологічний розрив між потребою у безперервному моніторингу відкритого вебпростору та відсутністю універсальних браузерних інструментів обґрунтовує доцільність розроблення інтегрованого розширення на базі фреймворку Svelte. Вибір даної технології дозволяє реалізувати високопродуктивний та енергоефективний механізм автоматизованого оцінювання надійності джерел безпосередньо в процесі серфінгу, забезпечуючи безперервний захист користувача без зміни його типової поведінкової моделі.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Svelte

Результати щорічного опитування розробників на платформі Stack Overflow підтверджують зростання популярності Svelte — відносно нового вебфреймворку з відкритим вихідним кодом, який наразі займає лідируючі позиції в рейтингах лояльності фахівців. Розроблений Річем Харрісом, Svelte знайшов широке практичне застосування в таких авторитетних організаціях, як «The New York Times», «IBM», «Square» та «Chess.com». На відміну від традиційних фреймворків, Svelte функціонує як компілятор, написаний на мові TypeScript, що переносить основне обчислювальне навантаження з етапу виконання в браузері на етап збирання проєкту, забезпечуючи максимальну продуктивність фінального продукту.

Фундаментальна відмінність Svelte від таких поширених рішень, як React або Vue, полягає у відмові від концепції віртуального DOM (Virtual DOM). В архітектурах React та Vue зміна стану ініціює повторне відтворення всієї програми у віртуальній моделі, після чого алгоритми порівняння (diffing) визначають мінімально необхідну кількість маніпуляцій зі справжнім DOM браузера. Хоча цей підхід є ефективнішим за пряме маніпулювання складними об'єктами DOM, він створює додаткові накладні витрати на обчислення в реальному часі. Розробникам часто доводиться вдаватися до ручної оптимізації (наприклад, через метод `shouldComponentUpdate`), щоб уникнути зайвих циклів рендерингу, що свідчить про певну недосконаліть даної моделі[4].

Svelte пропонує альтернативну парадигму: код, написаний у файлах формату `.svelte`, трансформується компілятором в абстрактне синтаксичне дерево, а згодом – у високоефективний чистий JavaScript-клас. Замість використання об'єктів DOM для збереження стану, фреймворк інтегрує початковий стан безпосередньо в розмітку, що суттєво прискорює первинне

завантаження сторінки. У процесі роботи застосунку Svelte відстежує зміни змінних у межах компонентів і точково оновлює лише відповідні фрагменти DOM, не вдаючись до перерендерингу всього вузла. Це не тільки підвищує швидкодію, але й, за свідченням експертів, зокрема Марка Фолькманна, надає розробнику більшу свободу у прямій взаємодії з DOM без ризику конфліктів із внутрішнім станом фреймворку.

Технологічні переваги архітектури Svelte підтверджуються об'єктивними бенчмарками, зокрема результатами тестування продуктивності Стефана Краузе. У ході порівняльного аналізу операцій із масивами даних (відтворення таблиці на 1000 рядків) Svelte демонструє показники, що перевершують результати таких інструментів, як Vue (v3.2), Angular (v12.0) та React (v17.0).

Таким чином, поєднання компіляційного підходу та відмови від віртуального DOM дозволяє створювати легковагові додатки з високим рівнем відгуку інтерфейсу, що є критично важливим для сучасних веборієнтованих систем.

Для об'єктивної оцінки технологічних переваг Svelte доцільно систематизувати дані порівняльних тестів (benchmarks), проведених Стефаном Краузе. У межах тестування вимірювалася швидкість виконання базових операцій над масивами даних (створення, оновлення та видалення 1000 рядків таблиці), а також обсяг споживаної пам’яті.

Далі за текстом (рис. 2.1) подано порівняльну характеристику продуктивності Svelte у зіставленні з іншими популярними фреймворками (метрики вказані у мілісекундах, де менше значення відповідає вищій ефективності).

Операція (DOM manipulation)	Svelte (v3.42.1)	Vue (v3.2.1)	React (v17.0.1)	Angular (v12.0.1)
Створення 1000 рядків	35.2 мс	41.3 мс	52.8 мс	48.6 мс
Оновлення всіх рядків	29.8 мс	34.5 мс	43.1 мс	41.2 мс
Часткове оновлення	3.5 мс	4.8 мс	6.2 мс	5.1 мс
Видалення рядка	15.6 мс	17.1 мс	21.4 мс	19.8 мс
Обмін рядків (swap)	8.4 мс	12.2 мс	15.6 мс	14.3 мс
Очищення таблиці	12.1 мс	13.5 мс	18.2 мс	16.5 мс
Використання пам'яті (MB)	4.1 МБ	6.8 МБ	9.4 МБ	10.2 МБ

Рис. 2. 1. Порівняльна характеристика продуктивності

Представлені дані дозволяють зробити кілька науково обґрунтованих висновків щодо архітектурної ефективності Svelte:

- *Швидкість маніпуляцій з DOM*: Svelte демонструє найнижчі показники затримки у всіх категоріях створення та модифікації елементів. Це є прямим наслідком відсутності віртуального DOM, що дозволяє оминати етапи порівняння дерев (diffing) та миттєво вносити зміни у справжній DOM браузера.
- *Оптимізація пам'яті*: Обсяг оперативної пам’яті, необхідний для функціонування Svelte-додатка, є суттєво меншим у порівнянні з конкурентами. Це зумовлено тим, що фреймворк не утримує в пам'яті складну структуру об'єктів віртуального дерева.
- *Ефективність на слабких пристроях*: Мінімальні накладні витрати (overhead) роблять Svelte оптимальним вибором для розроблення браузерних розширень та мобільних вебдодатків, де ресурси процесора та пам'яті можуть бути обмеженими.

Дані результати підтверджують гіпотезу про те, що перенесення логіки фреймворку на етап компіляції забезпечує суттєвий приріст швидкодії порівняно з традиційними runtime-фреймворками.

Доцільно також розглянути показники обсягу завантажуваних даних (Bundle Size), оскільки цей параметр безпосередньо корелює з часом ініціалізування браузерного розширення та загальною продуктивністю системи в умовах обмежених ресурсів [5].

Нижче (рис. 2.2) подано порівняльний аналіз розмірів бандлів для стандартного тестового застосунку (RealWorld App), що дозволяє оцінити «вартість» використання кожного фреймворку для кінцевого користувача.

Технологічний стек	Розмір (KB, стиснутий)	Коефіцієнт відносно Svelte
Svelte (v3)	**1.6 KB**	**1.0x**
Vue (v3)	32.4 KB	20.2x
React (v17) + ReactDOM	42.1 KB	26.3x
Angular (v12)	65.2 KB	40.7x

Рис. 2.2. Порівняльний аналіз

Обґрунтування вибору Svelte для браузерних розширень

На основі проведеного аналізу продуктивності та обсягу коду можна виділити наступні ключові аргументи на користь обраної технології:

- **Мінімальний Runtime-overhead:** Завдяки тому, що Svelte компілює компоненти у високоефективний імперативний код, фінальний бандл практично не містить самого фреймворку. Це критично важливо для браузерних розширень, які мають завантажуватися та виконуватися миттєво при відкритті кожної нової сторінки.
- **Висока швидкість реакції (TTI):** Показник Time to Interactive у Svelte-додатків є значно нижчим, ніж у конкурентів, що використовують Virtual DOM. Це забезпечує плавний UX-досвід при відображенні метаданих про надійність новин безпосередньо під час серфінгу.
- **Ефективність ініціалізування:** У контексті розроблення інструментів для боротьби з дезінформацією, де затримка в кілька секунд

може призвести до того, що користувач встигне засвоїти фейковий контент до його маркування, швидкість ініціалізування Svelte є вирішальною технічною перевагою.

Таким чином, вибір Svelte як технологічного фундаменту дослідження є науково та практично обґрунтованим, оскільки він забезпечує оптимальний баланс між швидкістю розроблення, продуктивністю виконання та мінімізацією споживання системних ресурсів.

Проведений аналіз сучасного стану інформаційного середовища підтвердив гіпотезу про критичне ускладнення механізмів дистрибуції цифрового контенту. Стрімка інфільтрація маніпулятивних повідомлень у масову свідомість, що базується на використанні алгоритмів персоналізації та психологічних тригерів, робить традиційні методи перевірки фактів малоефективними. Встановлено, що ключовою проблемою є висока динаміка оновлення дезінформації, яка випереджає можливості ручного експертного оцінювання, що актуалізує розроблення автоматизованих систем підтримки користувача.

Дослідження існуючих програмних аналогів (зокрема чат-ботів та вбудованих інструментів соціальних платформ) виявило їхню обмеженість межами конкретних екосистем та реактивний характер функціонування. На сьогодні спостерігається суттєвий технологічний розрив: при наявності розвинених засобів блокування реклами, ринок практично не пропонує універсальних, архітектурно легких браузерних рішень для верифікування автономних вебресурсів у режимі реального часу. Це підтверджує доцільність створення спеціалізованого програмного інструменту, інтегрованого безпосередньо в інтерфейс переглядача [5].

2.2. Node.js

У контексті розроблення програмного забезпечення для аналізу та верифікування цифрового контенту, вибір серверного середовища виконання

JavaScript - Node.js - є стратегічним рішенням, що базується на необхідності забезпечення високої швидкодії та масштабованості. Node.js не є мовою програмування, а виступає як середовище (runtime), що використовує високоефективний рушій V8 від Google для трансляції JavaScript у машинний код. Це дозволяє досягти продуктивності, порівнянної з рішеннями на базі низькорівневих мов, зберігаючи при цьому гнучкість розроблення.

Фундаментальною особливістю Node.js, що визначає його перевагу для даного дослідження, є подієво-орієнтована архітектура з неблокуючою моделлю введення-виведення. Традиційні серверні платформи створюють окремий потік для кожного нового запиту, що при великій кількості одночасних з’єднань призводить до надмірного споживання оперативної пам'яті. Натомість Node.js функціонує в межах одного головного потоку, використовуючи механізм Event Loop для делегування завдань операційній системі [6].

Для браузерного розширення, яке повинно одночасно обробляти HTTP-запити до декількох API (наприклад, для отримання рейтингів довіри, аналізу сертифікатів безпеки сайту та перевірки наявності джерела у «чорних списках»), така модель є критично важливою. Вона дозволяє системі не очікувати завершення одного запиту для ініціації іншого, що забезпечує асинхронність і миттєву реакцію інтерфейсу на дії користувача.

Невід’ємною частиною середовища є NPM (Node Package Manager) - найбільший у світі репозиторій програмних модулів. У межах розроблення інструменту боротьби з дезінформацією використання NPM дозволяє інтегрувати готові, протестовані спільнотою рішення для:

- Синтаксичного та лінгвістичного аналізу: використання бібліотек для оброблення природної мови (NLP), що допомагають виявляти маніпулятивні маркери в текстах.
- Автоматизації збирання проєкту: застосування інструментів типу Vite або Webpack, які під керівництвом Node.js оптимізують вихідний код, зменшуючи розмір розширення та підвищуючи швидкість його завантаження.

- Забезпечення безпеки: автоматизований моніторинг вразливостей у залежностях проєкту, що є першочерговим завданням для ПЗ, яке працює з персональними даними користувача в браузері.

Синергія з фреймворком Svelte

Node.js відіграє роль сполучної ланки між сирцевим кодом на Svelte та фінальним продуктом. Оскільки Svelte переміщує логіку фреймворку на етап компіляції, саме середовище Node.js забезпечує виконання цього процесу. Це дозволяє створювати «чистий» JavaScript, який не потребує важких клієнтських бібліотек для виконання. Така синергія мінімізує обчислювальне навантаження на клієнтську сторону (браузер), переносячи складні трансформації коду на етап розроблення.

ВИСНОВКИ ДО РОЗДІЛУ

Обґрунтовано, що оптимальною технологічною базою для реалізування такого інструменту є фреймворк Svelte. Порівняльний аналіз технічних метрик (швидкість маніпуляцій з DOM, обсяг споживаної пам’яті та розмір бандла) продемонстрував значну перевагу Svelte над runtime-фреймворками, такими як React, Vue та Angular. Завдяки компіляційному підходу та відмові від віртуального DOM, обрана технологія дозволяє забезпечити мінімальний час відгуку інтерфейсу, що є критичним для коректної роботи розширення без негативного впливу на користувацький досвід.

Результати даного розділу створюють теоретичне та технологічне підґрунтя для подальшого проектування архітектури та розроблення програмного забезпечення, що поєднуватиме високу продуктивність із ефективністю методів протидії інформаційним загрозам.

Імплементування Node.js у процес розроблення дозволяє вирішити комплекс завдань: від забезпечення асинхронної оброблення даних у реальному часі до створення стабільного та безпечного середовища для складання високопродуктивних компонентів системи. Це робить Node.js незамінним компонентом технологічного стеку для створення сучасних засобів медіабезпеки.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Налаштування технологій

Ефективна реалізація браузерного розширення для верифікування контенту потребує попереднього розгортання та коректного налаштування спеціалізованого стеку технологій. Підготовчий етап розроблення охоплює інсталяцію базових компонентів середовища виконання, вибір засобів проектування інтерфейсу та конфігурацію хмарної інфраструктури для збереження даних [7]. А саме:

1. Node.js – потрібний для використання npm це дозволить легко і просто встановлювати фреємворки і бібліотеки через термінал
2. Visual code – редактор коду для комфортної розроблення проекту (можна встановити інший аналог редактора)
3. Svelte – фреємворк за допомогою якого буде розроблятись логіка програми
4. Firebase – сервіс який дозволяє зберігати дані в реальному часі
5. Інше (Rollup, Babel, Yarn, Tailwind) – дані бібліотеки дозволять нам автоматизувати роботу, а саме: зібрати в один файл, оптимізувати код програми та стилізувати його.

Процес налаштування програмного середовища розпочинається з інсталяції Node.js. Для цього необхідно перейти на офіційний веб-ресурс розробників (https://nodejs.org) та ініціювати завантаження актуальної версії дистрибутива, скориставшись відповідним елементом інтерфейсу (рис. 3.1).

Рекомендується обирати версію з довгостроковою підтримкою (LTS), що гарантує стабільність роботи всіх залежних компонентів та бібліотек, які будуть інтегровані в проєкт на наступних етапах розроблення [8].

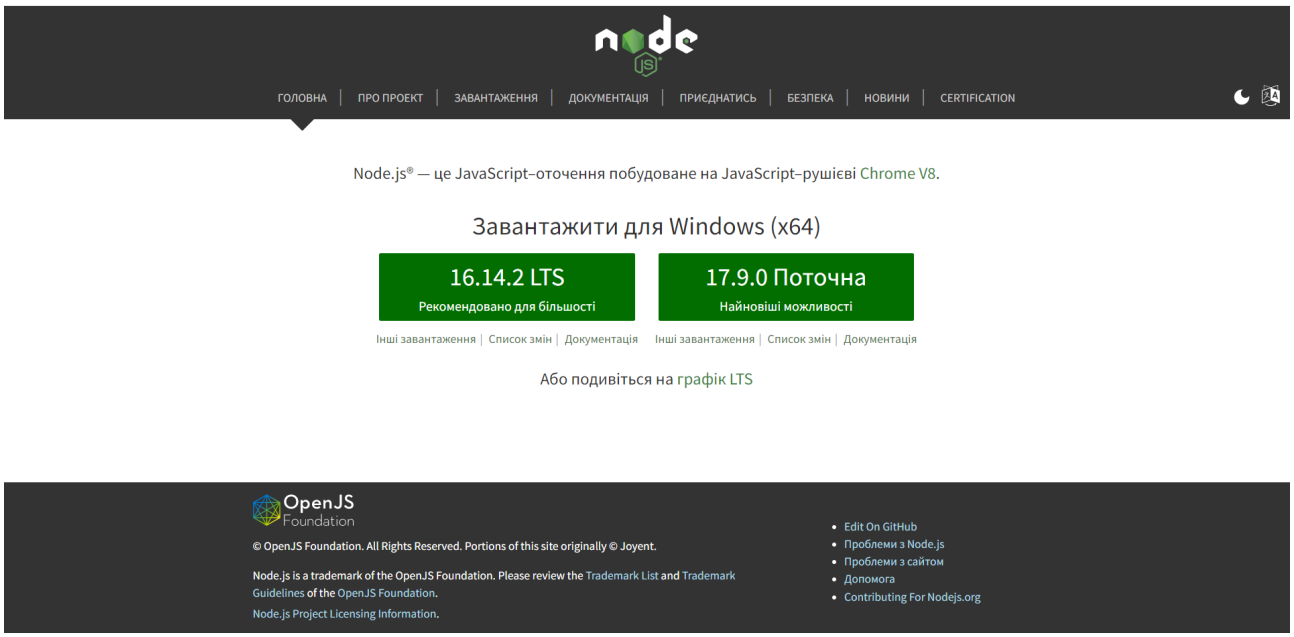


Рис 3.1. Офіційна сторінка Node.js

Після завершення завантаження інсталяційного пакета виконується процедура інсталяції середовища на локальну робочу станцію. З огляду на стандартний характер процесу встановлення та наявність інтуїтивно зрозумілого майстра налаштувань, детальний технічний опис кожного кроку цього етапу є недоцільним.

Наступним етапом є підготовка інтегрованого середовища розроблення (IDE), яке забезпечить інструментарій для написання та відлагодження програмного коду. Для цього необхідно перейти на офіційну сторінку завантаження за адресою https://code.visualstudio.com/ та завантажити дистрибутив, адаптований під відповідну операційну систему (рис. 3.2).

Вибір даного середовища зумовлений його широкими можливостями щодо розширення функціональності та нативною підтримкою мовних стандартів, що використовуються у проєкті.

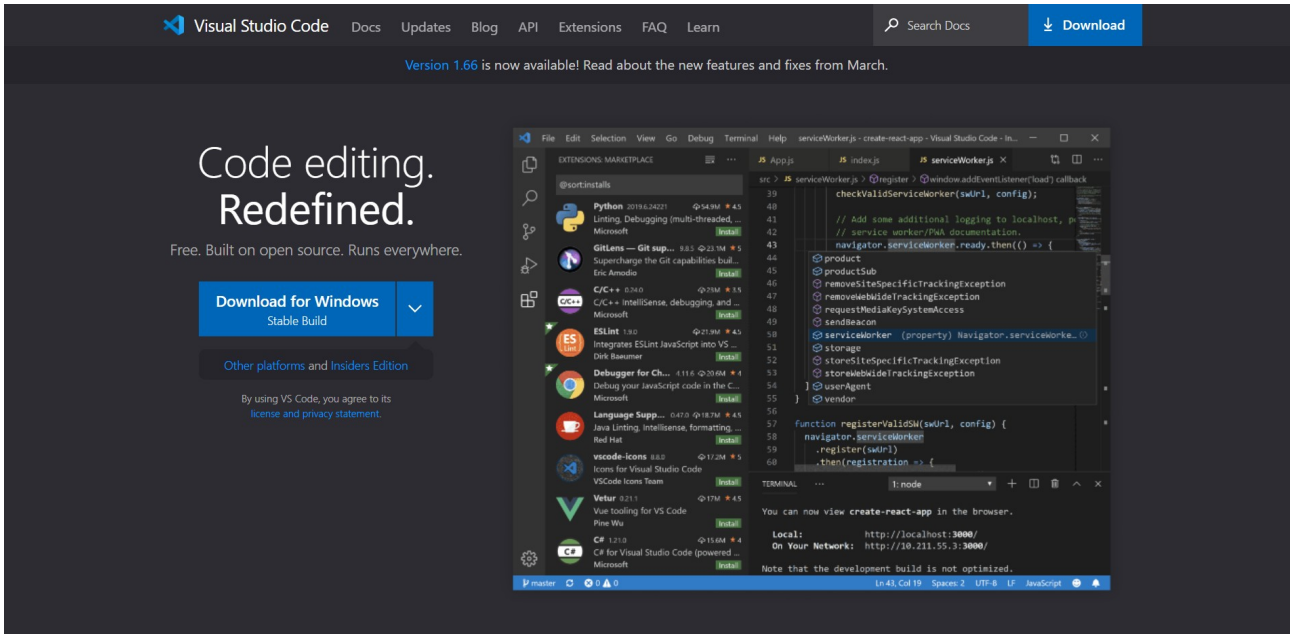


Рис 3.1.2 Офіційна сторінка Visual code

Після завершення інсталяції інтегрованого середовища розроблення та його запуску подальше розгортання проєкту здійснюється через вбудований інтерфейс термінала. Цей підхід дозволяє автоматизувати процес конфігурації базових компонентів системи за допомогою пакетних менеджерів.

Для розгортання початкової архітектури застосовується команда: *npx degit "kyrelldixon/svelte-tailwind-extension-boilerplate#main"*

Аналіз зазначеної операції підтверджує ініціалізацію комплексного шаблону, що автоматично інтегрує фреймворк Svelte, бібліотеку Tailwind CSS та допоміжний інструментарій збірки (рис. 3.3).

Оскільки платформа Firebase функціонує як незалежний сервіс, її інтеграція до складу проєкту потребує окремої інсталяції відповідного SDK. Для цього в терміналі виконується наступна команда: *npm install firebase*

Після завершення процесу розгортання та інсталяції всіх необхідних залежностей у робочому просторі IDE сформується відповідна файлова структура, що містить конфігураційні файли та директорії проєкту.

Така послідовність дій забезпечує створення стабільної технологічної бази, готової до подальшої реалізування логіки верифікування контенту та взаємодії з хмарною базою даних.

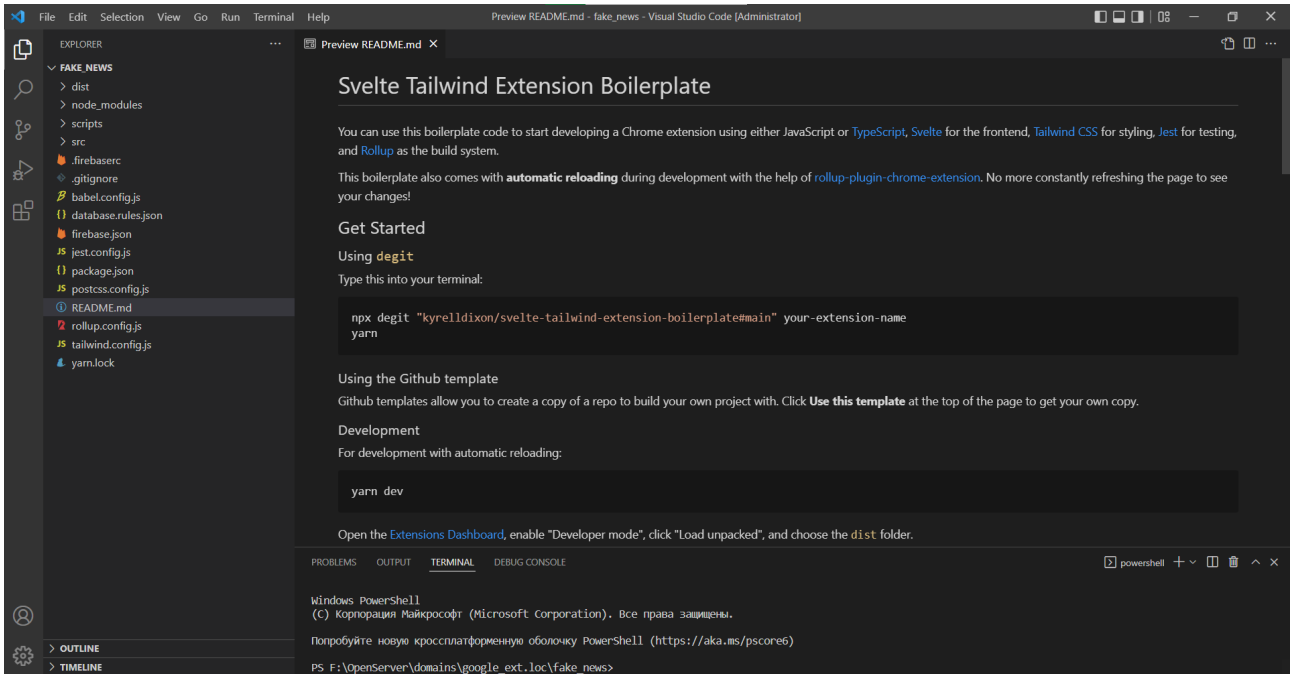


Рис 3.1. Встановлений набір Інструментів

Завершальним етапом підготовчих робіт є конфігурація хмарної інфраструктури на платформі Firebase, що передбачає створення бази даних та налаштування параметрів доступу для забезпечення подальшої взаємодії з програмним продуктом.

Процес ініціалізування розпочинається з авторизування на офіційному порталі сервісу, після чого необхідно ініціювати процедуру створення нового проєкту через консоль розробника (рис. 3.4).

На цьому етапі визначаються базові параметри проєкту, включаючи його ідентифікатор та регіональні налаштування, що є необхідним для розгортання NoSQL-бази даних Cloud Firestore та її інтеграції з клієнтською частиною браузерного розширення.

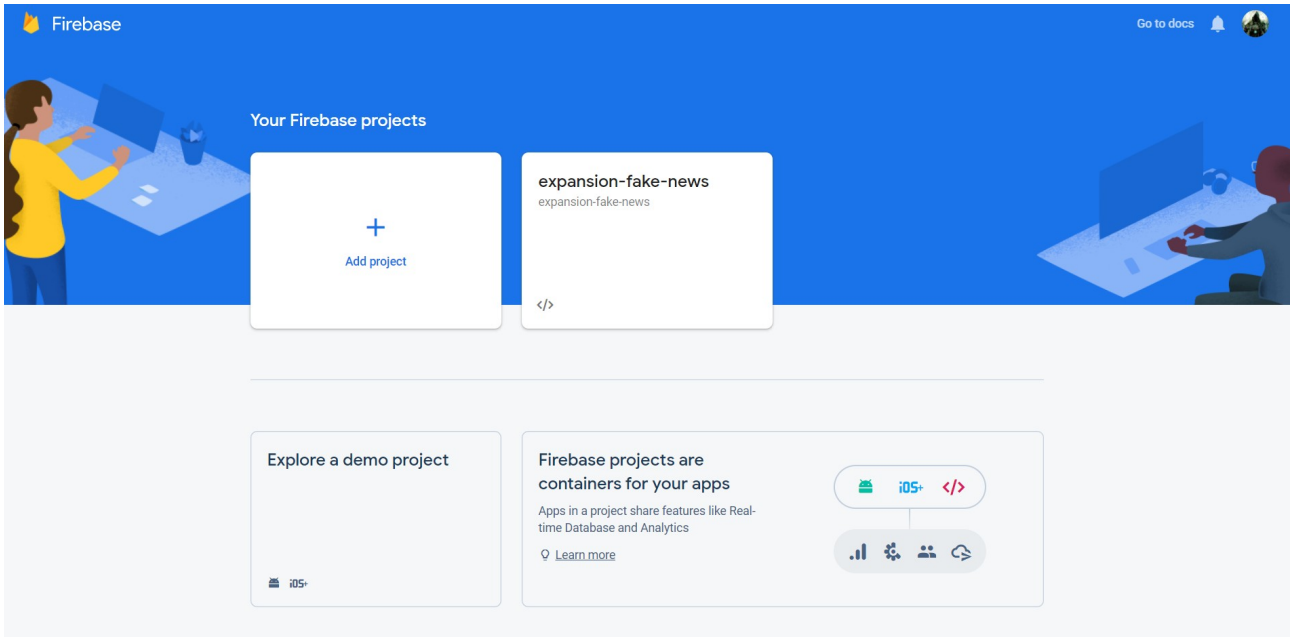


Рис 3.4. Firebase. Створення нового проекту

Ідентифікатор проєкту має бути уніфікований відповідно до назви програмного засобу, що розробляється. Після успішної ініціалізування проєкту виникає необхідність його детальної конфігурації для забезпечення авторизованого доступу до даних.

З цією метою здійснюється перехід до розділу «Authentication», де у вкладці «Sign-in method» встановлюються параметри автентифікування користувачів.

Серед переліку доступних механізмів авторизування необхідно активувати провайдер «Email/Password», що дозволить системі ідентифікувати користувачів за допомогою електронної пошти та пароля (рис. 3.5).

Дане налаштування є критично важливим для розмежування прав доступу та забезпечення персоналізації функцій браузерного розширення під час взаємодії з хмарною інфраструктурою.

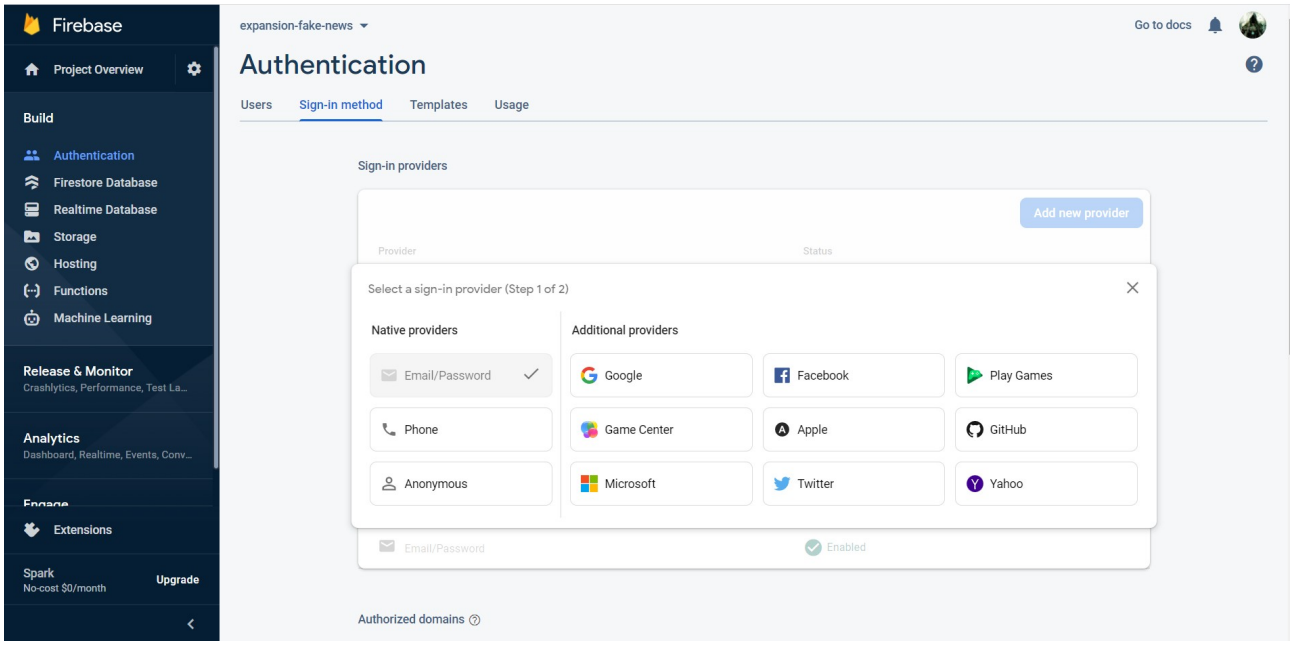


Рис 3.5. Firebase Authentication

Активування зазначеної функції дозволяє реалізувати механізм реєстрації користувачів у браузерному розширенні через адресу електронної пошти. Використання вбудованих інструментів Firebase суттєво оптимізує процес розроблення, оскільки платформа надає готову логіку автентифікування, що позбавляє необхідності проектувати та впроваджувати складні алгоритми ідентифікації на рівні клієнтської частини програми.

Наступним кроком є перехід до розділу «Realtime Database» для створення бази даних, функціональним призначенням якої є акумуляція та збереження відгуків користувачів.

Ці дані міститимуть оцінки вебресурсів, які відвідують суб'єкти інформаційної взаємодії, що дозволить формувати динамічний рейтинг довіри до джерел новин. Налаштування бази даних Realtime Database у консолі Firebase подано на рис. 3.6).

Такий підхід забезпечує обмін інформацією між користувачами в режимі реального часу, що є ключовим елементом системи колективної оцінки достовірності контенту [9, 10].

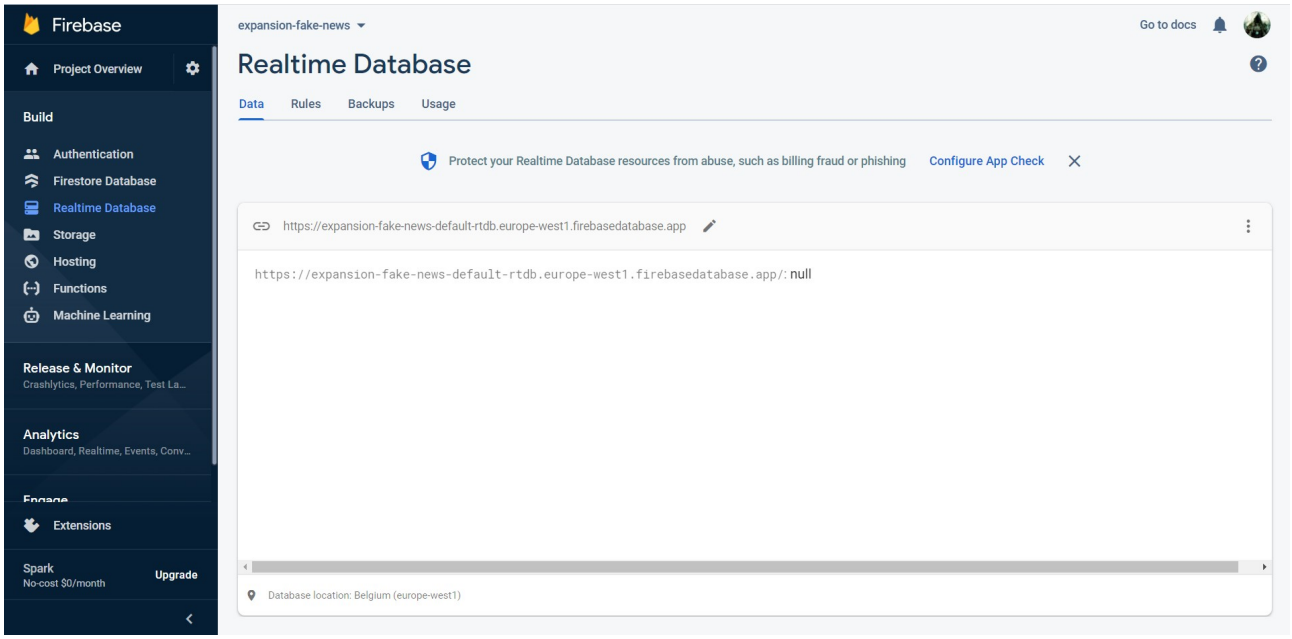


Рис 3.6. Firebase Realtime Database Налаштування бази даних Realtime Database у консолі Firebase

Безпосередньо перед початком етапу програмної реалізування необхідно звернутися до параметрів конфігурації проєкту для отримання об’єктів доступу та ключів приватності, адаптованих для вебтехнологій. Ці дані є критично важливими для встановлення захищеного з’єднання між клієнтською частиною браузерного розширення та хмарним сервісом за допомогою прикладного програмного інтерфейсу (API).

Приклад отримання конфігураційних даних та API-ключів у консолі Firebase подано на рис. 3.7.

Належне налаштування цих параметрів гарантує коректну ідентифікацію застосунку серверами Firebase та забезпечує можливість безпечного передавання й отримання даних, що є необхідною умовою для функціонування системи верифікування в режимі реального часу.

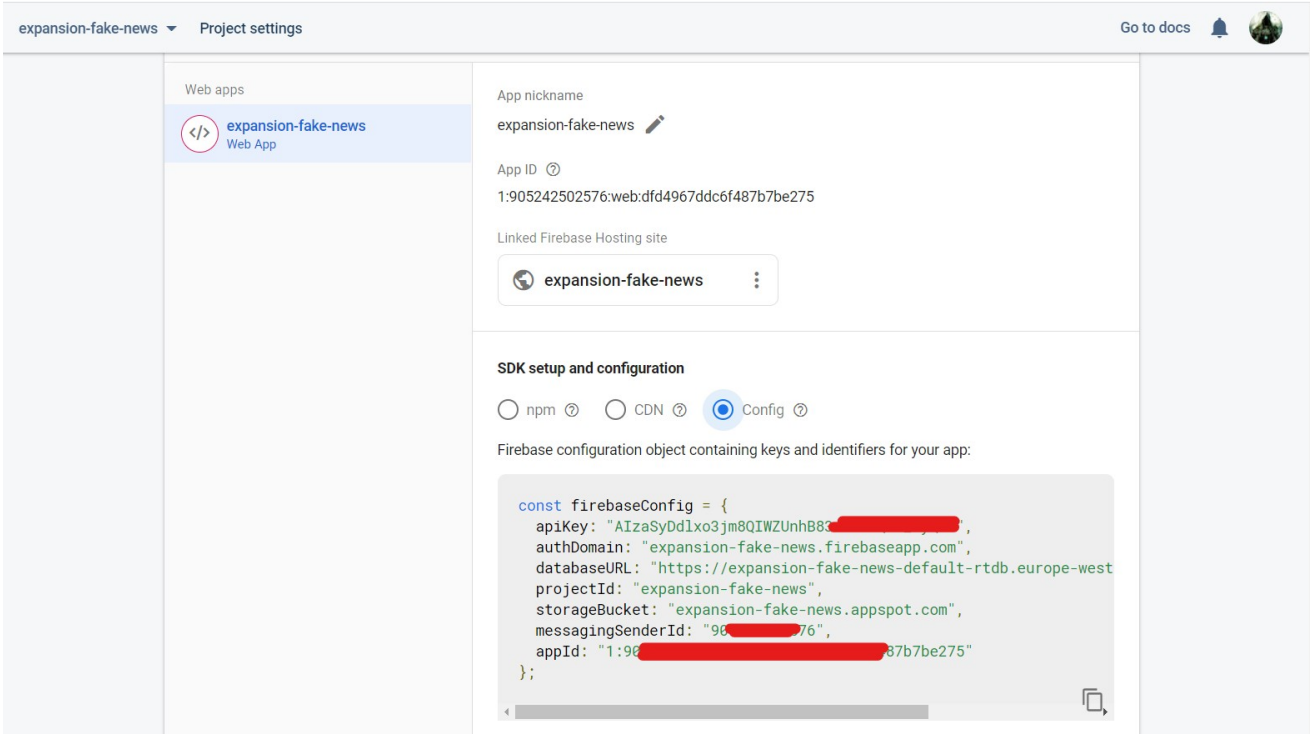


Рис 3.7. Отримання конфігураційних даних та API-ключів у консолі Firebase

Таким чином, завершено підготовчий етап, та сформовано середовище, що об'єднує локальні інструменти програмування (Node.js, Visual Studio Code, Svelte) та хмарні сервіси (Firebase), створює необхідні умови для переходу до безпосередньої програмної реалізування функціональних модулів системи.

3.2. Розроблення логіки розширення

Початковим етапом розроблення є створення головного конфігураційного файлу, який забезпечує централізоване керування параметрами функціонування розширення.

Процес створення головного конфігураційного файлу проілюстровано рис. 3.8.

Ідентифікатор даного файлу «manifest.json».

src > {} manifest.json > {} icons > 128

```
{
  "name": "Fake news",
  "version": "0.0.1",
  "description": "DEVELOPMENT duplam",
  "icons": {"128": "./logo2.png"},
  "background": { "scripts": ["background/index.js"] },
  "content_scripts": [
    {
      "js": ["content/index.js"],
      "matches": ["https://*/*", "http://*/*"]
    }
  ],
  "browser_action": { "default_popup": "popup/index.html" },
  "permissions": [
    "storage"
  ]
}

```

Рис 3.8. Процес створення головного конфігураційного файлу Manifest

Програмний код, імплементований у даному файлі, детермінує ключові метадані та операційні параметри розширення. Нижче подано деталізацію основних дескрипторів:

- **manifest_version**: Визначає специфікацію формату маніфесту, що використовується для забезпечення сумісності з платформою браузера.
- **name**: Ідентифікатор програмного продукту (назва розширення).
- **description**: Текстова анотація, що розкриває функціональне призначення та особливості розширення.
- **version**: Поточний номер ітерації (версія) програмного забезпечення.
- **icons**: Набір графічних ресурсів, що використовуються для візуалізації розширення в інтерфейсі керування браузера.
- **background**: Скрипти, призначені для виконання в ізольованому фоновому середовищі незалежно від життєвого циклу веб-сторінок.
- **content_scripts**: Сценарії, що мають безпосередній доступ до об'єктної моделі документа (DOM) активних вкладок для модифікації вмісту.

- **browser_action**: Модуль взаємодії з користувачем через інтерфейс браузера:
    - **default_popup**: Основний HTML-документ, що ініціалізується при активування іконки розширення.
    - **default_icon**: Графічний символ розширення, що відтворюється на панелі інструментів (toolbar).
- **permissions**: Декларація привілеїв та доступів до API браузера, необхідних для коректного функціонування системних компонентів.

У нашому середовищі відкриємо папку «src» а у ній папку «popup» в якій створимо наступні файли за правилами фреємворку Svelte. На рис. 3.9 подано приклад архітектури popup сторінки.

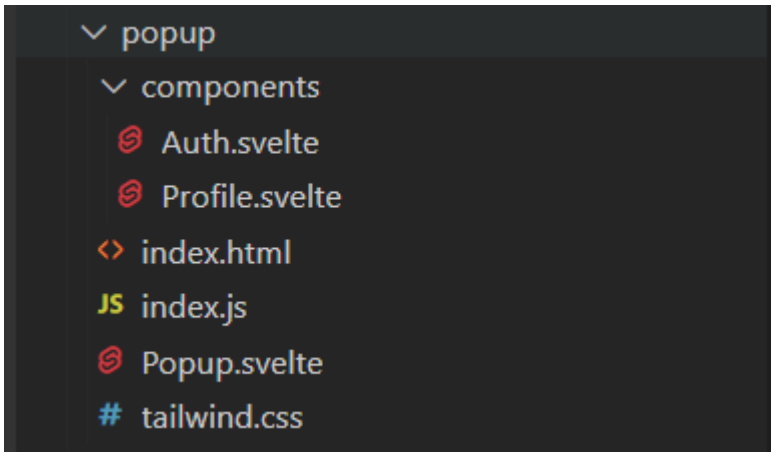


Рис. 3.9. Архітертура popup сторінки

Процес розроблення інтерфейсу випливаючого вікна (popup) передбачає імплементацію функціоналу авторизування та реєстрації користувачів. Використання фреймворку Svelte дозволяє реалізувати компонентно-орієнтований підхід, що забезпечує декомпозицію програмної логіки на окремі модульні одиниці [11]. Приклад декомпозицію програмної логіки на окремі модульні одиниці Popup файл index.html подано на рис. 3.10.

Рис 3.10. Декомпозиція програмної логіки на окремі модульні одиниці Popup файл index.html

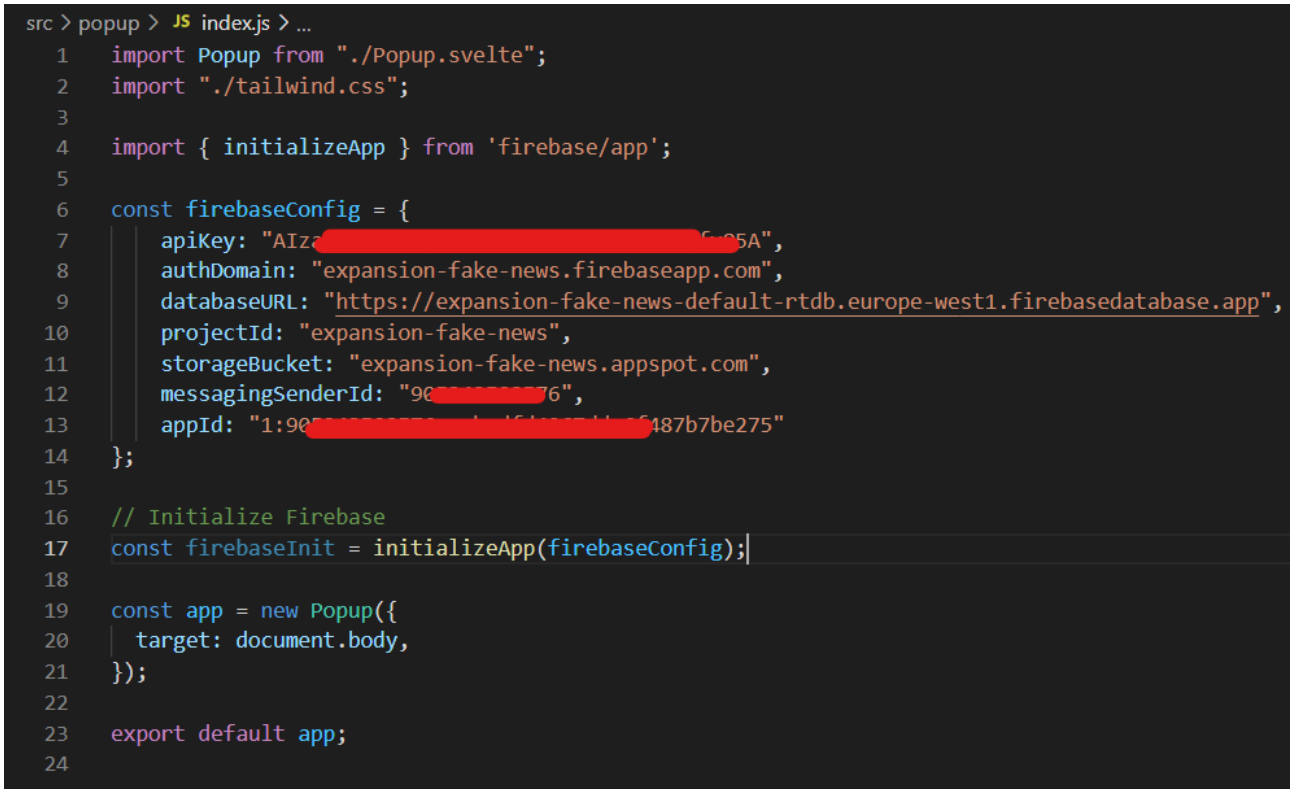
Відзначений документ містить стандартне HTML-розмічування, яке слугує каркасом для ініціалізування клієнтської частини додатка. Архітектура файлу передбачає інтеграцію ключових залежностей та системних компонентів:

Технічні характеристики структури index.html

• Точка входу JavaScript: Під’єднання основного скрипту, що забезпечує імпорт та рендеринг кореневого компонента Popup.svelte.

• Стилізація: Інтеграція бібліотеки Tailwind CSS для забезпечення адаптивного та компонентного дизайну інтерфейсу за допомогою утилітарних класів.

• Інфраструктурний шар: Під’єднання та ініціалізування сервісів Firebase, що дозволяє реалізувати механізми автентифікування та взаємодії з хмарною базою даних у реальному часі Popup файл index.js, що відтворено на рис. 3.11.



```
src > popup > JS index.js > ...
1   import Popup from "./Popup.svelte";
2   import "./tailwind.css";
3
4   import { initializeApp } from 'firebase/app';
5
6   const firebaseConfig = {
7       apiKey: "AIza[redacted]5A",
8       authDomain: "expansion-fake-news.firebaseapp.com",
9       databaseURL: "https://expansion-fake-news-default-rtdb.europe-west1.firebasedatabase.app",
10      projectId: "expansion-fake-news",
11      storageBucket: "expansion-fake-news.appspot.com",
12      messagingSenderId: "90[redacted]76",
13      appId: "1:90[redacted]487b7be275"
14  };
15
16  // Initialize Firebase
17  const firebaseInit = initializeApp(firebaseConfig);
18
19  const app = new Popup({
20    target: document.body,
21  });
22
23  export default app;
24
```

Рис 3.11. Popup файл index.js механізми автентифікування та взаємодії з хмарною базою даних у реальному часі Popup файл index.js

У компоненті Popup.svelte реалізовано логіку керування станом інтерфейсу та взаємодії із зовнішніми сервісами.

Архітектура компонента передбачає імпорт функціональних модулів Profile та Auth, а також методів бібліотеки firebase/database для маніпуляції даними у хмарному сховищі.

На рис. 3.12 подано приклад імпорт функціональних модулів Profile та Auth, а також методів бібліотеки firebase/database для маніпуляції даними у хмарному сховищі, файл Popup.svelte javascript код.

src > popup > Popup.svelte > ...

```
<script>
    import Profile from "./components/Profile.svelte";
    import Auth from "./components/Auth.svelte";
    import { getDatabase, ref, set } from "firebase/database";

    const db = getDatabase();
    let user;

    chrome.storage.local.get(['user'], (data) => {
        user = data.user;
    });

    function storageSetUser (data) {
        user = data.detail;

        chrome.storage.local.set({user: user}, () => {});

        set(ref(db, 'users/' + user.uid), {
            online: true
        });
    }

    function storageClearUser (data) {
        set(ref(db, 'users/' + user.uid), {});

        chrome.storage.local.clear();
        user = null
    }
</script>
```

Рис 3.12. приклад імпорт функціональних модулів Profile та Auth, а також методів бібліотеки firebase/database для маніпуляції даними у хмарному сховищі, файл Popup.svelte javascript код.

У межах компонента Popup.svelte реалізовано структуру HTML-шаблону, який складається з інформаційного заголовка (header) із назвою розширення та блоку динамічного рендерингу контенту. Для керування станом інтерфейсу застосовано механізм умовної логіки, що базується на поточному статусі автентифікування користувача, файл Popup.svelte html. (рис. 3.13).

```
<main class="w-64 flex flex-wrap content-between bg-gray-100 p-3">

    <div class="w-full flex justify-between items-center mb-3">

        <a href="index.html" class="flex items-center">
            <img src="/logo2.png" class="w-12 h-10" alt="">
            <spam class="font-bold text-red-700 text-xl ml-2">FakeNews</spam>
        </a>

    </div>

    <div class="w-full bg-white rounded-md px-3 py-2">
        {#if user}
            <Profile
                on:logaut={storageClearUser}
                {user}
            />
        {:else}
            <Auth
                on:user={storageSetUser}
            />
        {/if}
    </div>

</main>
```

Рис 3.13. Механізм умовної логіки, що базується на поточному статусі автентифікування користувача, файл Popup.svelte html.

Імплементація компонента автентифікування передбачає інтеграцію методів авторизування та реєстрації з бібліотеки Firebase SDK, а також використання стандартного інструментарію createEventDispatcher для забезпечення передачі подій до батьківського компонента Popup.svelte (рис. 3.14, файл Auth.svelte javascript код).

Програмна логіка компонента базується на двох функціях оброблення, що ініціюються під час активування відповідних елементів інтерфейсу («Вхід» та «Реєстрація»).

Дані функції інтегровані в HTML-структуру форми введення, що дозволяє синхронізувати дії користувача з процесами оброблення облікових даних на рівні додатка (рис. 3.15, файл Auth.svelte html).

src > popup > components > Auth.svelte > ...

```
<script>
    import { createEventDispatcher } from "svelte";
    import { getAuth, createUserWithEmailAndPassword, signInWithEmailAndPassword  } from "firebase/auth";
    const dispatche = createEventDispatcher();
    const auth = getAuth();
    let email;
    let password;
    let errorData;
    function signInUser () {
        signInWithEmailAndPassword(auth, email, password)
        .then((userCredential) => {
            // Signed in
            dispatche('user', userCredential.user);
            errorData = null;
        })
        .catch((error) => {
            errorData = {
                code : error.code,
                message : error.message
            }
        });
    }
    function createUser () {
        createUserWithEmailAndPassword(auth, email, password)
        .then((userCredential) => {
            // Signed in
            dispatche('user', userCredential.user);
            errorData = null;
        })
        .catch((error) => {
            errorData = {
                code : error.code,
                message : error.message
            }
        });
    }
</script>
```

Рис 3.14. Файл Auth.svelte javascript код

```
<div id="form_block" class="w-full">
    {#key errorData}
        {#if errorData}
            <div id="error" class="text-red-600">
                {errorData.code} <br> {errorData.message}
            </div>
        {/if}
    {/key}

    <input
        bind:value={email}
        id="email"
        type="email"
        class="w-full border px-4 py-2 my-2"
        placeholder="Email">

    <input
        bind:value={password}
        id="password"
        type="password"
        class="w-full border px-4 py-2 my-2"
        placeholder="Password">

    <div class="grid grid-cols-2 gap-3 font-bold mt-3">
        <button
            on:click={signInUser}
            class="w-full bg-blue-600 hover:bg-blue-500 transform active:scale-95 text-white rounded-lg px-4 py-2">
            Вхід
        </button>
        <button
            on:click={createUser}
            class="w-full bg-green-600 hover:bg-green-500 transform active:scale-95 text-white rounded-lg px-4 py-2">
            Реєстрація
        </button>
    </div>
</div>
```

Рис 3.15. Файл Auth.svelte html

Реалізація компонента Profile.svelte вирізняється мінімалістичною архітектурою, оскільки його основне функціональне призначення полягає у забезпеченні інтерфейсу для завершення сеансу користувача [12].

Програмна логіка компонента базується на використанні стандартного механізму createEventDispatcher, який дозволяє передавати події до батьківського файлу. Зокрема, у компоненті імплементовано функцію ініціації виходу з облікового запису; ця функція лише тригерує подію, тоді як безпосередня обробка операції розлогінювання та деструктуризації сесії здійснюється на рівні вищого компонента Popup.svelte (рис. 3.16, файл Profile.svelte).

src > popup > components > Profile.svelte > div.flex.justify-end

```
<script>
    import { createEventDispatcher } from "svelte";

    export let user;

    const dispatche = createEventDispatcher();

    function logaut () {
        dispatche('logaut');
    }
</script>

<div class="text-center text-md border-b-2 pb-2 mb-2">
    {user.email}
</div>

<div class="flex justify-end">
    <button
        on:click={logaut}
        class="transform bg-red-600 hover:bg-red-500 active:scale-95 text-white px-4 py-2">
        Logout
    </button>
</div>
```

Рис 3.16. Файл Profile.svelte

У компоненті Profile.svelte реалізовано відтворення електронної пошти користувача та елемента керування для виходу із системи. Після інсталяції розширення активування його іконки в інтерфейсі браузера ініціює відкриття вікна з формою автентифікування. Приклад Вікно розширення з формою вводу даних подано на рис. 3.17.

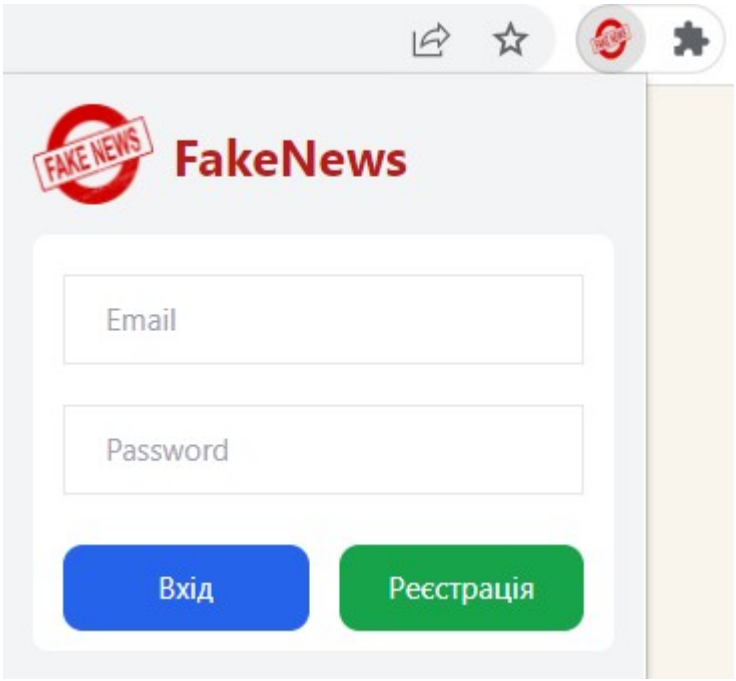


Рис 3.17. Вікно розширення з формою вводу даних

Після введення облікових даних інтерфейс автоматично, без перезавантаження сторінки, відтворює вікно профілю користувача. При цьому відповідні записи успішно синхронізуються та зберігаються у базі даних Firebase.

Приклад вікна з створеним профілем подано на рис. 3.18.

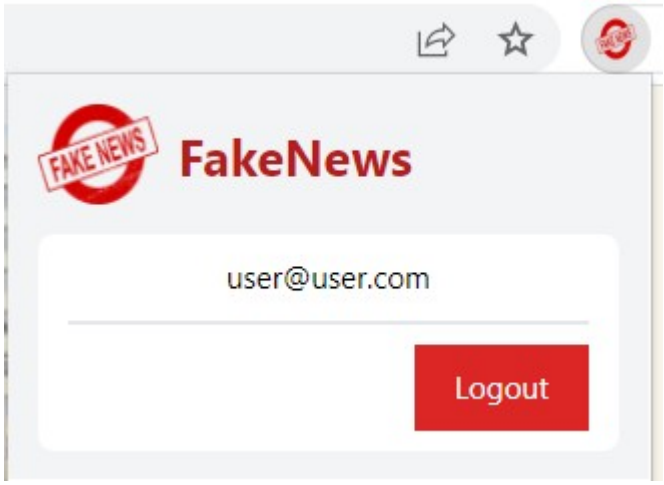


Рис 3.18. Вікно з профілем

Приклад Firebase БД авторизування подано на рис. 3.19.

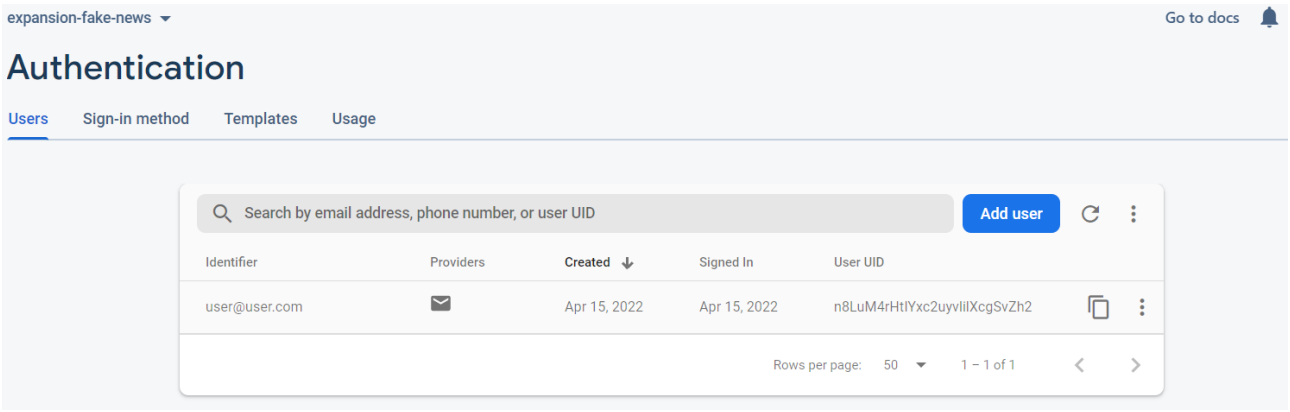


Рис 3.19. Firebase БД Авторизування

Після успішного реалізування модуля автентифікування наступним етапом є розроблення інтерфейсу системи відгуків. Для імплементації даного функціоналу в директорії content створюється набір файлів, архітектура яких частково дублює структуру раніше розроблених компонентів.

Архітертуру content інтерфейсу подано на рис. 3.20.

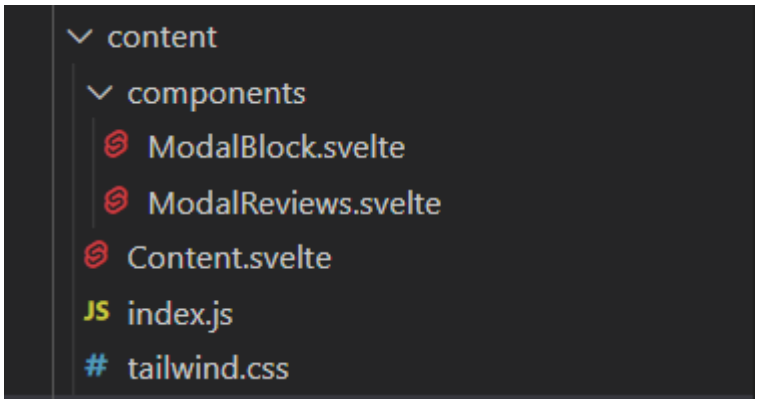


Рис 3.20. Архітертура content інтерфейсу

Файл index.js ідентичний за структурою попередньому, за винятком ініціалізування кореневого компонента Content.svelte. Основна логіка даного модуля зосереджена на маніпуляціях з DOM-елементами та взаємодії з базою даних. Першочергово імплементується під'єднання двох функціональних компонентів: перший призначений для блокування інтерфейсу за наявності негативного рейтингу ресурсу, другий — для відтворення аналітичної інформації про сторінку з можливістю додавання відгуків та коментарів.

Також здійснюється інтеграція Firebase SDK та ініціалізування змінних дефолтними значеннями (placeholder objects), що дозволяє запобігти

виникненню помилок звернення до властивостей об’єкта під час асинхронного очікування відповіді від бази даних (рис. 3.21, компонент «Content.svelte» початкові дані)

src > content > Content.svelte > script > ref

```
<script>
    import ModalBlock from "./components/ModalBlock.svelte";
    import ModalReviews from "./components/ModalReviews.svelte";
    import { getDatabase, ref, onValue } from "firebase/database";

    const db = getDatabase();

    // Створюємо юзера з початковими даними для запобіганя помилок при звернені до не існуючого ключа юзера
    let user = {
        uid: 'none',
        online: false
    };
    // Аналонічно створюємо початкові дані для сторінки
    let dataPage = {
            like: 0,
            dislike: 0,
            countReviews: 0,
        };
```

Рис 3.21. Компонент «Content.svelte» початкові дані

Наступним етапом здійснюється парсинг доменного імені; за допомогою регулярних виразів символи крапки та косої риски замінюються на нижнє підкреслення. Це обумовлено синтаксичними обмеженнями Firebase, де URL-адреса виступає унікальним ключем у таблиці domains (рис. 3.22, компонент «Content.svelte» отримання даних з бази даних).

Після формування ключа виконується зчитування даних користувача з поточної сесії та встановлення підписки на оновлення бази даних для забезпечення реактивності інтерфейсу без перезавантаження сторінки. Функціональну частину завершують методи керування станом модальних вікон: закриття блокувального екрана та активування інтерфейсу відгуків [13].

У HTML-шаблоні реалізовано логіку умовного рендерингу: якщо кількість відгуків перевищує порогове значення (10 одиниць) і частка негативних оцінок є домінуючою, система автоматично активує компонент блокування доступу (рис. 3.23, компонент «Content.svelte» html розмічування).

```
20      // Формуємо посилання яку будемо збережете в бд як ключ дла сторінки
21      let hostname = window.location.hostname.replace(/\./g, "_");
22      let pathname = window.location.pathname.replace(/[\/\.]/g, "__");
23      const urlSite = hostname + '' + `${pathname == '__' ? '' : pathname}`;
24
25      // Получаємо дані користувача з сесії
26      chrome.storage.local.get(['user'], (data) => {
27          if(data.user) user = data.user;
28      });
29      $: {
30          // Підписуємось на бд щоб в реальгому часі відслідковувати зміни для таблиці "users"
31          onValue(ref(db, 'users/' + user.uid), (snapshot) => {
32              const data = snapshot.val();
33              if (data) {
34                  user.online = true;
35              }
36              else{
37                  user.online = false;
38              }
39          });
40      }
41      // Підписуємось на бд щоб в реальгому часі відслідковувати зміни для таблиці "domains"
42      onValue(ref(db, 'domains/' + urlSite), (snapshot) => {
43          const data = snapshot.val();
44          if (data) dataPage = data;
45      });
46      // Метод який викликає вункції з дочірнього компонента
47      let childReviews;
48      function showReviews() {
49          childReviews.toggle();
50      }
51      // Перевіряємо чи користувач зажав вказану ними комбінацію клавіш
52      function handleKeydown(event) {
53          if (event.ctrlKey  &&  event.altKey  &&  event.key === "f") {
54              showReviews();
55          }
56      }
```

Рис 3.22. Компонент «Content.svelte» отримання даних з бази даних

```
61  <div id="moduleFakeNews">
62      {#if dataPage.countReviews > 10 && dataPage.dislike > dataPage.like}
63          <ModalBlock
64              on:showReviews={showReviews}
65          />
66      {/if}
67
68      <ModalReviews
69          bind:this={childReviews}
70          {user}
71          {dataPage}
72          {urlSite}
73      />
74  </div>
```

Рис 3.23.Компонент «Content.svelte» html розмічування

Далі розробимо програмну логіку компонента ModalBlock.svelte. Даний модуль забезпечує функціонування інтерфейсу блокування, обробляючи стани

відтворення та взаємодію з користувачем у разі виявлення критичного рейтингу ресурсу. (рис. 3.24 компонент «ModalBlock.svelte» javascript код) та (рис. 3.25 компонент «ModalBlock.svelte» html розмічування).

src > content > components > ModalBlock.svelte > script

```
<script>
    import { createEventDispatcher } from "svelte";

    document.querySelector('body').classList.add('overflow-hidden')

    function GoBack() {
        history.go(-1)
    }

    function CloseModelBlock () {
        document.querySelector('body').classList.remove('overflow-hidden')
        document.querySelector('#FakeNews_ModelBlock').remove()
    }

    const dispatche = createEventDispatcher();

    function hendleShowReviews() {
        dispatche('showReviews');
        CloseModelBlock();
    }

</script>
```

Рис 3.24. Компонент «ModalBlock.svelte» javascript код

```
<div id="FakeNews_ModelBlock" class="fixed inset-0 bg-red-600 z-[2147483646] flex justify-center items-center">

    <div class="text-white px-10 py-5">

        <div class="text-3xl text-center mb-5">
            <strong>Обережно!</strong>
        </div>

        <div class="text-xl mb-5">
            <p>За думкою користувачів Інформація яка тут знаходиться є неправдивою, будьте обережні при перегляді даної сторінки)</
        </div>

        <div class="flex justify-center">
            <button
                class="bg-custom-main hover:bg-custom-accent text-white cursor-pointer px-10 py-5"
                on:click={GoBack}>
                Повернутись на зад
            </button>

            <button
                class="bg-custom-main hover:bg-custom-accent text-white cursor-pointer px-10 py-5"
                on:click={CloseModelBlock}>
                Переглянути
            </button>

            <button
                class="bg-custom-main hover:bg-custom-accent text-white cursor-pointer px-10 py-5"
                on:click={hendleShowReviews}>
                Відгуки
            </button>
        </div>

    </div>

</div>
```

Рис 3.25. Компонент «ModalBlock.svelte» html розмічування

У межах компонента розроблено три функції для оброблення подій натискання відповідних клавіш, інтегрованих у HTML-шаблон вікна блокування. Розмічування також містить інформаційний блок із текстом попередження для користувача. Візуалізування підсумкового інтерфейсу подано на рис. 3.26, Вікно Блокування.

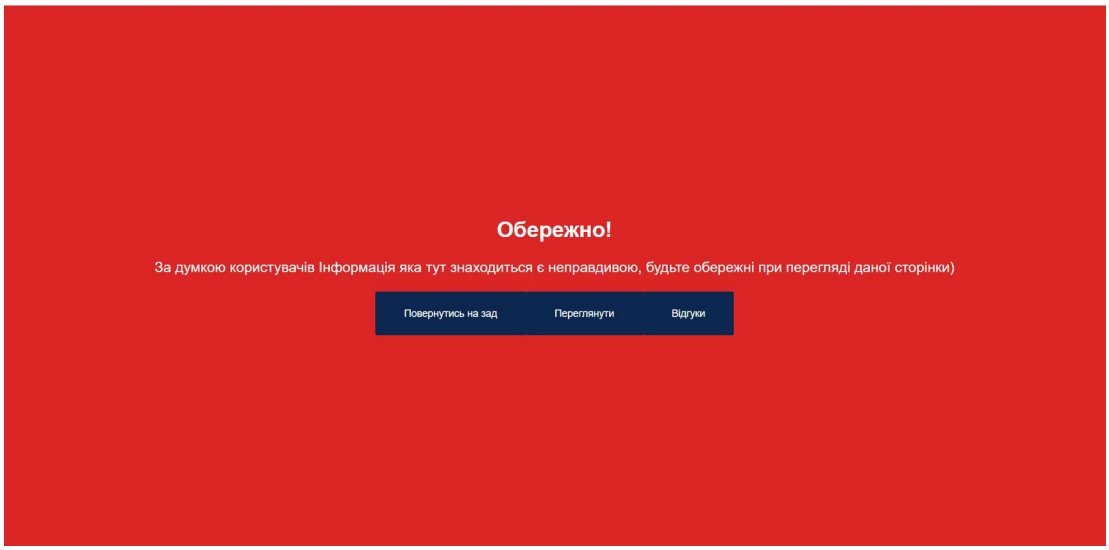


Рис 3.26. Вікно Блокування

Останнім етапом є розроблення компонента ModalReviews.svelte. Згідно з архітектурним задумом, активування даного вікна здійснюється або через інтерфейс блокування, або за допомогою клавіатурної комбінації Ctrl + Alt + F. Використання гарячих клавіш дозволяє відмовитися від статичних елементів керування, що забезпечує безперешкодний перегляд контенту сторінки [14].

Графічний інтерфейс компонента структурований наступним чином:

• ***Верхня панель***: Відтворення поточної URL-адреси, загальної кількості відгуків та статистичних даних щодо вподобань (лайків/дизлайків).

• ***Центральна частина***: Стрічка повідомлень користувачів, що включає оцінку (позитивну або негативну), опціональний текстовий коментар та ідентифікатор автора (email). Це подано на рис. 3.27, Інтерфейс вікна Відгуки]

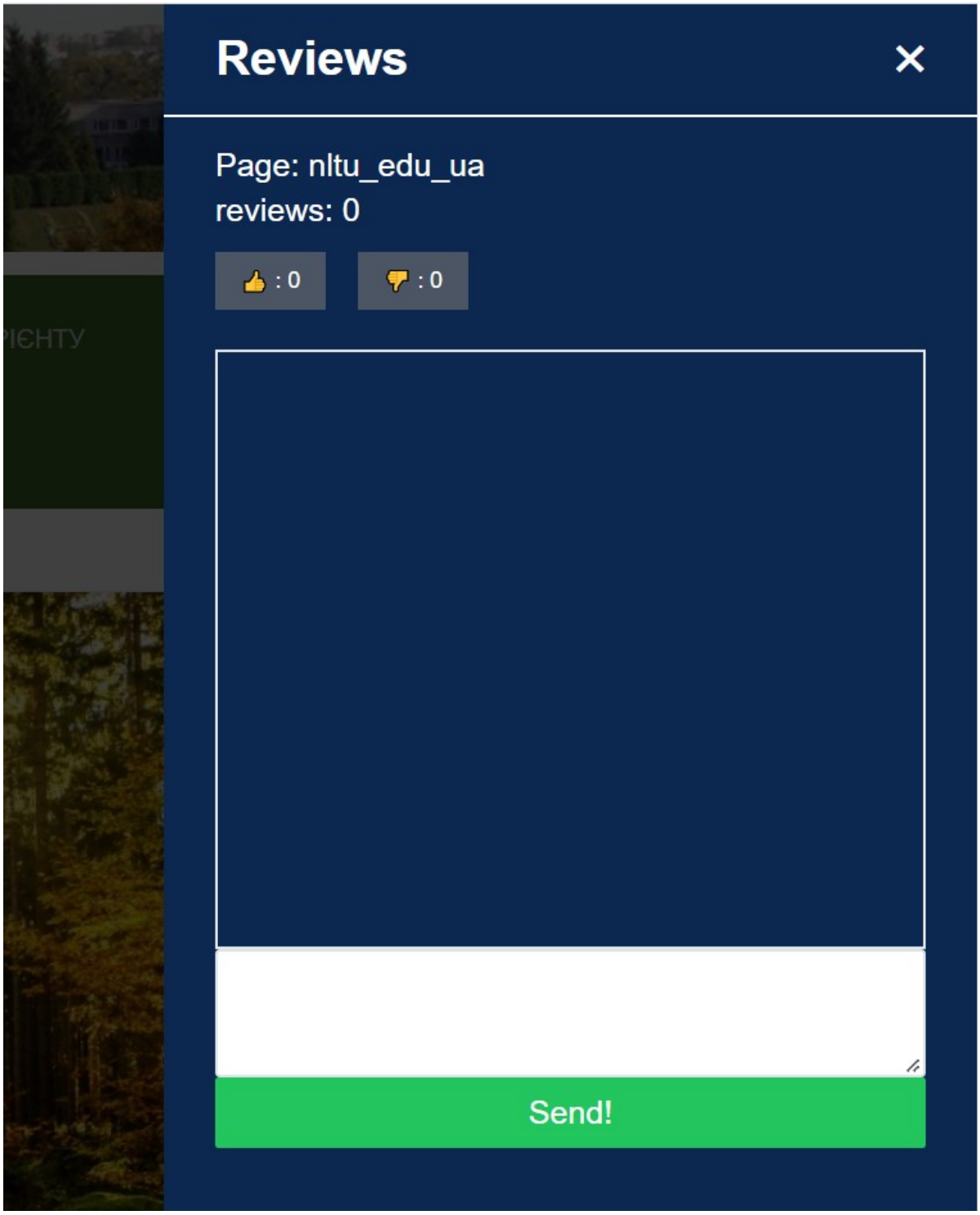


Рис 3.27. Інтерфейс вікна Відгуки

Після завершення візуальної частини необхідно імплементувати програмну логіку для динамічної взаємодії з Firebase. Використання директиви export у Svelte забезпечує отримання зовнішніх даних компонентом ModalReviews.svelte від батьківського модуля Content.svelte, зокрема ідентифікатора користувача, параметрів сторінки та сформованих посилань.

Окрім змінних, за допомогою ключового слова export оголошено функції плавного відтворення та приховування інтерфейсу. Це дозволяє ініціювати анімаційні ефекти та керувати станом модального вікна за межами його внутрішньої логіки. На рис. 3.28 подано приклад реалізування компоненти «ModalReviews.svelte» дані з можливістю «export».

src > content > components > ModalReviews.svelte > script

```
<script>
    import { getDatabase, ref, set, onValue } from "firebase/database";
    const db = getDatabase();

    export let user;
    export let dataPage;
    export let urlSite;

    let blockBlur;
    let FakeNews_ModelReviews;
    export function toggle() {
        blockBlur.classList.toggle("bg-black")
        blockBlur.classList.toggle("bg-opacity-75")
        blockBlur.classList.toggle("pointer-events-none")
        FakeNews_ModelReviews.classList.toggle("translate-x-full")
    }
```

Рис 3.28. Компонент «ModalReviews.svelte» дані з можливістю «export»

Наступним етапом є ініціалізування локальних змінних із визначенням їхніх початкових типів даних: логічні значення (boolean) для оцінок «Подобається» та «Не подобається», текстове поле для повідомлення, масив (array) для переліку всіх відгуків та об’єкт (object) для зберігання даних поточного користувача [15].

Надалі впроваджується метод моніторингу даних, який перевіряє наявність попереднього відгуку авторизованого користувача для поточної сторінки. У разі виявлення відповідних записів, їхні значення присвоюються вищезгаданим локальним змінним. Аналогічний алгоритм застосовується для отримання та синхронізації загального масиву відгуків, що належать до ідентифікованого посилання (рис. 3.29, компонент «ModalReviews.svelte» сполучення даних з БД).

```
    let like = false;
    let dislike = false;
    let message = '';
    let reviews = [];
    let reviewUser;

    if(user) {
        // Перевіряємо чи є в базі даних відгук наш відгук по сторінці яку ми відкрили
        onValue(ref(db, '/reviews/' + urlSite + '/' + user.uid), (snapshot) => {
            const data = snapshot.val();
            if (data) {
                like = data.like;
                dislike = data.dislike;
                reviewUser = data;
            }
        });
    };

    // Получаємо усі відгуки які залишині по сторінці яку ми відкрили
    onValue(ref(db, '/reviews/' + urlSite), (snapshot) => {
        console.log
        if (snapshot) {
            let arr = [];
            snapshot.forEach((childSnapshot) => {
                const childKey = childSnapshot.key;
                const childData = childSnapshot.val();
                arr = [...arr, childData];
            });
            reviews = arr;
        }
    });
```

Рис 3.29. Компонент «ModalReviews.svelte» сполучення даних з БД

Реалізовано метод для внесення відгуку та супровідного коментаря (за його наявності) до бази даних (рис. 3.30). Дана функція забезпечує валідацію введених параметрів та їх подальшу синхронізацію з відповідним вузлом у структурі Firebase.

```
function setUserReviews(){
    set(ref(db, '/reviews/' + urlSite + '/' + user.uid), {
        like: like,
        dislike: dislike,
        message: message == '' && reviewUser ? reviewUser.message : message,
        email: user.email
    });

    message = '';
}
```

Рис 3.30. Компонент «ModalReviews.svelte» метод для запису відгука в БД

Заключні два методи призначені для коректної оброблення вибору користувача між опціями «Подобається» та «Не подобається». Програмна логіка цих функцій забезпечує точність підрахунку голосів, гарантуючи фіксацію лише одного унікального відгуку для кожної сесії, незалежно від обраної полярності оцінки (рис. 3.31, компонент «ModalReviews.svelte» логіка відгуків).

```
50      function checkLike() {
51          like = !like;
52
53          let coustReviews = dataPage.countReviews;
54          if(dislike == false){
55              coustReviews = like ? dataPage.countReviews + 1 : dataPage.countReviews - 1;
56          }
57
58          set(ref(db, 'domains/' + urlSite), {
59              like: like ? dataPage.like + 1 : dataPage.like - 1,
60              dislike: dislike ? dataPage.dislike - 1 : dataPage.dislike,
61              countReviews: coustReviews
62          });
63
64          dislike = false;
65          setUserReviews()
66      }
67
68      function checkDislike() {
69          dislike = !dislike;
70
71          let coustReviews = dataPage.countReviews;
72          if(like == false){
73              coustReviews = dislike ? dataPage.countReviews + 1 : dataPage.countReviews - 1;
74          }
75
76          set(ref(db, 'domains/' + urlSite), {
77              like: like ? dataPage.like - 1 : dataPage.like,
78              dislike: dislike ? dataPage.dislike + 1 : dataPage.dislike - 1,
79              countReviews: coustReviews
80          });
81
82          like = false;
83          setUserReviews()
84      }
```

Рис 3.31. Компонент «ModalReviews.svelte» логіка відгуків

Заключним етапом є реалізація механізму виведення отриманих із бази даних відомостей у користувацький інтерфейс. Дана логіка забезпечує динамічне відтворення актуальної інформації про відгуки та рейтинги безпосередньо у відповідних блоках розмічування [16].

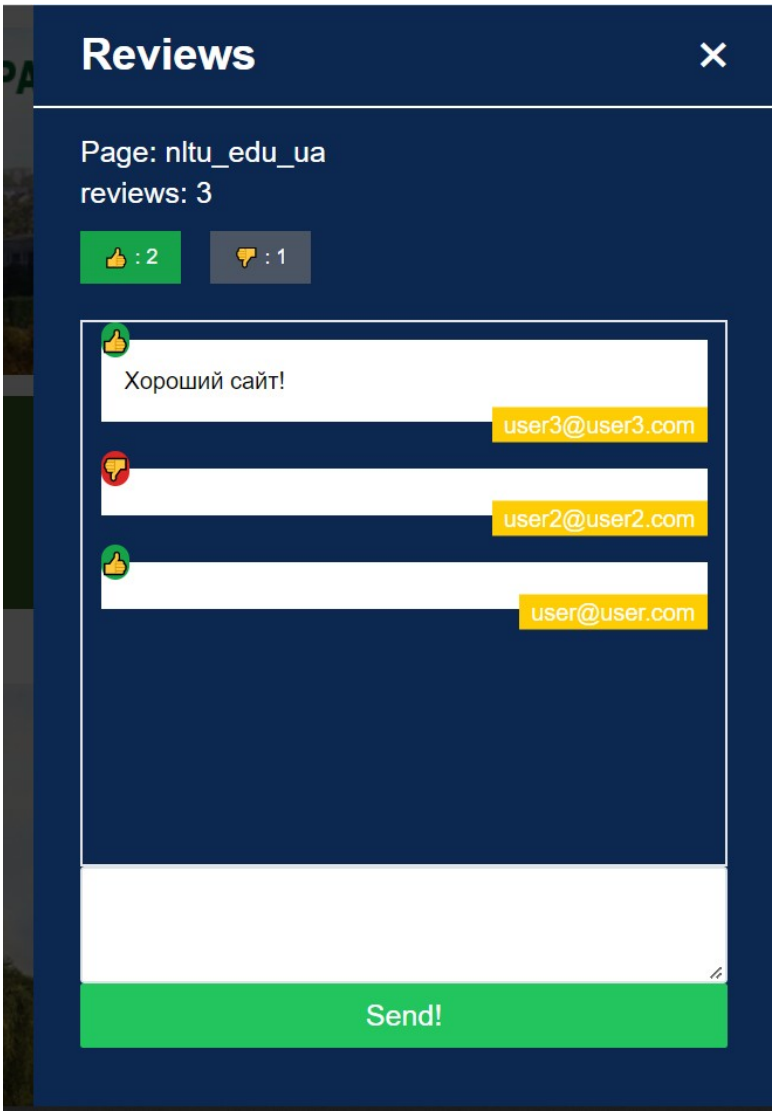


Рис 3.32. Результат роботи

ВИСНОВКИ ДО РОЗДІЛУ

На основі проведеного дослідження було встановлено, що початковим етапом розроблення є створення головного конфігураційного файлу, який забезпечує централізоване керування параметрами функціонування розширення, що разом із обранням фреймворку Svelte дозволило досягти високої оптимізації логіки та інтерфейсу.

Вибір даного технологічного стеку обґрунтований результатами порівняльного аналізу продуктивності, згідно з якими Svelte демонструє найкращі показники серед сучасних рішень.

ВИСНОВКИ

Завершальним етапом науково-практичної роботи стало комплексне проєктування та імплементація програмного забезпечення, спрямованого на верифікацію контенту та мінімізацію впливу дезінформації на користувачів мережі. Розроблене рішення базується на використанні фреймворку Svelte для забезпечення високої реактивності користувацького інтерфейсу, що у поєднанні з Node.js дозволило створити гнучку та масштабовану архітектуру додатка. Роль централізованого сховища даних відіграє хмарна платформа Firebase Realtime Database, яка забезпечує синхронізацію інформації в режимі реального часу, що є критично важливим для актуальності даних про репутацію інформаційних ресурсів.

Функціональна архітектура системи охоплює надійний механізм реєстрації та автентифікування користувачів, гарантуючи ідентифікацію суб'єктів взаємодії. Особливу увагу приділено розробці інтерактивного інтерфейсу збору фідбеку, який дозволяє користувачам висловлювати оціночні судження щодо джерел інформації та коментувати їх. Додатково впроваджено модуль адміністративної модерації, що забезпечує контроль над якістю контенту та дотриманням етичних норм дискусії, мінімізуючи ймовірність маніпуляцій та поширення спаму в системі відгуків.

Процес розроблення було реалізовано з використанням сучасного інструментарію автоматизації, що підвищило продуктивність та якість кінцевого коду. Управління залежностями здійснювалося засобами Yarn, тоді як інструмент збірки Rollup забезпечив ефективну агрегацію модулів в оптимізовані файлові бандли. Завдяки застосуванню транспілятора Babel було досягнуто високого рівня сумісності додатка з різними середовищами виконання, а використання фреймворку Tailwind CSS сприяло розробці адаптивного та ергономічного візуального стилю. Сукупність цих технологічних рішень дозволила створити інструмент, що не лише відповідає

сучасним стандартам веброзроблення, але й ефективно виконує поставлені завдання щодо підвищення медіаграмотності користувачів шляхом надання верифікованої інформації про джерела контенту.

Розроблене програмне забезпечення є комплексним інструментом, що поєднує сучасні вебтехнології для розв'язання актуальної проблеми верифікування контенту. Використання реактивного фреймворку Svelte у синергії з хмарною інфраструктурою Firebase дозволило створити високопродуктивну систему з архітектурою, що забезпечує оновлення даних у реальному часі та надійну автентифікацію користувачів. Застосування автоматизованих засобів збірки та оптимізації коду гарантує стабільну роботу додатка в різних браузерних середовищах. Таким чином, реалізований проєкт закладає надійний технологічний фундамент для подальшого впровадження інтелектуальних методів аналізу інформації та масштабування функціоналу, спрямованого на зміцнення інформаційної безпеки та підвищення рівня медіаграмотності суспільства.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Springer S. Node.js: The Comprehensive Guide. — Boston : Rheinwerk Publishing, 2022. — 834 p.

2. Alessandro Segala: Svelte 3 Up and Running: A fast-paced introductory guide to building high-performance web applications with SvelteJS – Packt Publishing Ltd, 2020

3. Robin Wieruch: The Road to Firebase: Your journey to master Firebase in JavaScript – Independently published (June 7, 2021)

4. Harris R. Svelte: Cybernetically enhanced web apps: documentation. https://svelte.dev/docs.

5. Firebase Realtime Database: Store and sync data in real time : documentation / Google Cloud. URL: https://firebase.google.com/docs/database.

6. Node.js v20.x documentation / Node.js Foundation. https://nodejs.org/docs/latest/api/.

7. Tailwind CSS: Utility-first CSS framework : documentation / Tailwind Labs Inc. https://tailwindcss.com/docs.

8. Rollup.js: The JavaScript module bundler : documentation. https://rollupjs.org/guide/en/.

9. Babel: The compiler for next generation JavaScript : documentation. https://babeljs.io/docs/en/.

10. Yarn: Dependency Management : documentation. https://yarnpkg.com/getting-started.

11. ECMAScript® 2026 Language Specification / ECMA International. https://www.ecma-international.org/publications-and-standards/standards/ecma-262/.

12. Vaughan A. Real-World Svelte: Build fast, scalable, and reactive web applications. Birmingham : Packt Publishing, 2023. 312 p.

13. Chinnathambi K. Learning Svelte: Buy, build, and deploy your first Svelte app. London : Addison-Wesley Professional, 2022. 256 p.

14. Wardle C., Derakhshan H. Information Disorder: Toward an interdisciplinary framework for research and policymaking. Strasbourg : Council of Europe, 2017. 102 p.

15. Lazer D. J. The science of fake news. *Science*. 2018. Vol. 359, Issue 6380. P. 1094–1096.

16. Web Content Accessibility Guidelines (WCAG) 2.2 / W3C. https://www.w3.org/TR/WCAG22/.

ДОДАТКИ

Додаток А.

Програмний код для реалізування розширення в браузері для протидії з фейковими новинами

Popup.svelte

```
<script>
    import Profile from "./components/Profile.svelte";
    import Auth from "./components/Auth.svelte";
    import { getDatabase, ref, set } from
"firebase/database";
    const db = getDatabase();
    let user;
    chrome.storage.local.get(['user'], (data) => {
        user = data.user;
    });
    function storageSetUser (data) {
        user = data.detail;
        chrome.storage.local.set({user: user}, () =>
{});
        set(ref(db, 'users/' + user.uid), {
            online: true
        });    }
    function storageClearUser (data) {
        set(ref(db, 'users/' + user.uid), {});
                chrome.storage.local.clear();
        user = null    }
</script>
<main class="w-64 flex flex-wrap content-between bg-
gray-100 p-3">
    <div class="w-full flex justify-between items-
center mb-3">
```

```
        <a href="index.html" class="flex items-center">
            <img src="/logo2.png" class="w-12 h-10"
alt="">
            <spam class="font-bold text-red-700 text-xl
ml-2">FakeNews</spam>
        </a>    </div>
    <div class="w-full bg-white rounded-md px-3 py-2">
        {#if user}
            <Profile
                on:logaut={storageClearUser}
                {user}            />
        {:else}
            <Auth on:user={storageSetUser}    />
        {/if}    </div></main>
```

Auth.svelte

```
<script>
    import { createEventDispatcher } from "svelte";
    import { getAuth, createUserWithEmailAndPassword,
signInWithEmailAndPassword  } from "firebase/auth";
    const dispatche = createEventDispatcher();
    const auth = getAuth();
    let email;
    let password;
    let errorData;
    function signInUser () {
        signInWithEmailAndPassword(auth, email,
password)
        .then((userCredential) => {
            // Signed in
            dispatche('user', userCredential.user);
```

```
            errorData = null;        })
        .catch((error) => {
            errorData = {
                code : error.code,
                message :
error.message       }        });    }
    function createUser () {
        createUserWithEmailAndPassword(auth, email,
password)
        .then((userCredential) => {
            // Signed in
            dispatche('user', userCredential.user);
            errorData = null;        })
        .catch((error) => {
            errorData = {
                code : error.code,
                message : error.message     }    });
}
</script>
<div id="form_block" class="w-full">
    {#key errorData}
        {#if errorData}
            <div id="error" class="text-red-600">
                {errorData.code} <br>
{errorData.message}
            </div>        {/if}    {/key}
    <input
        bind:value={email}
        id="email"
        type="email"
```

```
        class="w-full border px-4 py-2 my-2"
        placeholder="Email">
    <input
        bind:value={password}
        id="password"
        type="password"
        class="w-full border px-4 py-2 my-2"
        placeholder="Password">
    <div class="grid grid-cols-2 gap-3 font-bold mt-3">
        <button on:click={signInUser}
            class="w-full bg-blue-600 hover:bg-blue-500
transform active:scale-95 text-white rounded-lg px-4
py-2">            Вхід        </button>        <button
            on:click={createUser}
            class="w-full bg-green-600 hover:bg-
green-500 transform active:scale-95 text-white rounded-
lg px-4 py-2">
            Реєстрація
        </button>
    </div>
</div>
```

Profile.svelte

```
<script>
    import { createEventDispatcher } from "svelte";
    export let user;
    const dispatche = createEventDispatcher();
    function logaut () {   dispatche('logaut');    }
</script>
<div class="text-center text-md border-b-2 pb-2 mb-2">
    {user.email}</div>
```

```
<div class="flex justify-end">
    <button on:click={logaut}
        class="transform bg-red-600 hover:bg-red-500
active:scale-95 text-white px-4 py-2">
        Logout    </button>  </div>
```

Content.svelte

```
<script>
    import ModalBlock from
"./components/ModalBlock.svelte";
    import ModalReviews from
"./components/ModalReviews.svelte";
    import { getDatabase, ref, onValue } from
"firebase/database";
    const db = getDatabase();
    // Створюємо юзера з початковими даними для
запобіганя помилок при звернені до не існуючого ключа
юзера
    let user = {        uid: 'none',        online:
false    };
    // Аналонічно створюємо початкові дані для сторінки
    let dataPage = {            like: 0,
dislike: 0,            countReviews: 0,        };
    // Формуємо посилання яку будемо збережете в бд як
ключ дла сторінки
    let hostname =
window.location.hostname.replace(/\./g, "_");
    let pathname =
window.location.pathname.replace(/[\/\.]/g, "__");
    const urlSite = hostname + '' + `${pathname == '__'
? '' : pathname}`;


    // Получаємо дані користувача з сесії
```

```
    chrome.storage.local.get(['user'], (data) =>
{    if(data.user) user = data.user;    });

    $: {// Підписуємось на бд щоб в реальгому часі
відслідковувати зміни для таблиці "users"

        onValue(ref(db, 'users/' + user.uid),
(snapshot) => {

            const data = snapshot.val();

            if (data) {   user.online = true;      }

            else{   user.online =
false;      }        });    }

       // Підписуємось на бд щоб в реальгому часі
відслідковувати зміни для таблиці "domains"

    onValue(ref(db, 'domains/' + urlSite), (snapshot)
=> {

        const data = snapshot.val();

        if (data) dataPage = data;    });

    // Метод який викликає вункції з дочірнього
компонента

    let childReviews;

    function showReviews() {childReviews.toggle();}

    // Перевіряємо чи користувач зажав вказану ними
комбінацію клавіш

    function handleKeydown(event) {

        if (event.ctrlKey  &&  event.altKey  &&
event.key === "f") {

            showReviews();        }    }</script>

<svelte:window on:keydown={handleKeydown}/>

  <div id="moduleFakeNews">

    {#if dataPage.countReviews > 10 && dataPage.dislike
> dataPage.like}

        <ModalBlock  on:showReviews={showReviews}
/>
```

```
    {/if}    <ModalReviews  bind:this={childReviews}
{user} {dataPage} {urlSite}/> </div>

ModalBlock.svelte <script>

    import { createEventDispatcher } from "svelte";


document.querySelector('body').classList.add('overflow-
hidden')

    function GoBack() {        history.go(-1)    }

    function CloseModelBlock () {


document.querySelector('body').classList.remove('overfl
ow-hidden')


document.querySelector('#FakeNews_ModelBlock').remove()
}

    const dispatche = createEventDispatcher();

    function hendleShowReviews() {

        dispatche('showReviews');

        CloseModelBlock();    }</script>

<div id="FakeNews_ModelBlock" class="fixed inset-0 bg-
red-600 z-[2147483646] flex justify-center items-
center">

    <div class="text-white px-10 py-5">

        <div class="text-3xl text-center mb-5">

            <strong>Обережно!</strong>        </div>

        <div class="text-xl mb-5">

            <p>За думкою користувачів Інформація яка
тут знаходиться є неправдивою, будьте обережні при
перегляді даної сторінки)</p>        </div>

        <div class="flex justify-center">

            <button

                class="bg-custom-main hover:bg-custom-
accent text-white cursor-pointer px-10 py-5"
```

```
                on:click={GoBack}>
                Повернутись на зад
            </button>
             <button
                class="bg-custom-main hover:bg-custom-
accent text-white cursor-pointer px-10 py-5"
                on:click={CloseModelBlock}>
                Переглянути
            </button>
            <button
                class="bg-custom-main hover:bg-custom-
accent text-white cursor-pointer px-10 py-5"
                on:click={hendleShowReviews}>
                Відгуки
            </button></div>  </div></div>
```

ModalReviews.svelte

```
<script>
    import { getDatabase, ref, set, onValue } from
"firebase/database";
    const db = getDatabase();
    export let user;
    export let dataPage;
    export let urlSite;
    let blockBlur;
    let FakeNews_ModelReviews;
    export function toggle() {
        blockBlur.classList.toggle("bg-black")
        blockBlur.classList.toggle("bg-opacity-75")
        blockBlur.classList.toggle("pointer-events-
none")
```

```
FakeNews_ModelReviews.classList.toggle("translate-x-
full")    }

    let like = false;    let dislike = false;    let
message = '';    let reviews = [];    let reviewUser;

    $:{ if(user) {// Перевіряємо чи є в базі даних
відгук наш відгук по сторінці яку ми відкрили

            onValue(ref(db, '/reviews/' + urlSite + '/'
+ user.uid), (snapshot) => { const data =
snapshot.val();  if (data) { like = data.like;  dislike
= data.dislike; reviewUser = data;} }); }; }

     // Получаємо усі відгуки які залишині по сторінці
яку ми відкрили

    onValue(ref(db, '/reviews/' + urlSite), (snapshot)
=> {

        console.log        if (snapshot)
{            let arr = [];

            snapshot.forEach((childSnapshot) => {

                const childKey = childSnapshot.key;

                const childData = childSnapshot.val();

                arr = [...arr, childData];  });
reviews = arr;  }   });
```

```
// Initialize Firebase
const firebaseInit = initializeApp(firebaseConfig);


const app = new Content({
  target: document.body,
});
export default app;
```