Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Розробка розширення веб-браузера для автоматичного виправлення помилок при наборі тексті із використанням Java Script

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121 «Інженерія програмного Забезпечення»
(шифр і назва спеціальності)

(підпис) Буряк Д. В. (прізвище та ініціали)

Керівник (підпис) Мудрик І. Я. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю. М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль 2025

АНОТАЦІЯ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2025, Сторінок 54, таблиць 1, рисунків 16, презентація.

Тема: Розробка розширення для браузера для автоматичного виправлення помилок у тексту із використанням NodeJS та React.

У кваліфікаційній роботі описано процес розробки інструменту, призначеного для автоматичного виправлення тексту, набраного українською мовою з активною англійською розкладкою клавіатури. Реалізація була виконана у вигляді веб-додатку з подальшим переходом у формат розширення для браузера. Основна мета - підвищити зручність введення тексту шляхом автоматичного перетворення некоректного введення в коректну українську версію. В роботі проаналізовано подібні рішення та визначено їх переваги та недоліки. Розроблено структуру системи, реалізовано інтерфейс користувача за допомогою React та створено функціонал транслітерації на основі мапування символів. Серверна частина побудована за допомогою Node.js. Наведено приклади функціонування розробленого рішення та можливості його подальшого розвитку.

Ключові слова: транслітерація, мовна розкладка, веб-додаток, React, Node.js, автоматична корекція тексту, розширення для браузера.

ABSTRACT

Bachelor's thesis. Ternopil National Technical University named after Ivan Puluj, Department of Software Engineering, specialty 121
“Software Engineering”. TNTU, 2025, Pages 54, tables 1, figures 16, presentation.

Topic: Development of a browser extension for automatic error correction in text using NodeJS and React.

The qualification work describes the process of developing a tool designed to automatically correct text typed in Ukrainian with an active English keyboard layout. The implementation was done in the form of a web application with the subsequent transition to the format of a browser extension. The main goal is to increase the convenience of text input by automatically converting incorrect input into the correct Ukrainian version. The paper analyzes similar solutions and identifies their advantages and disadvantages. The system structure is developed, the user interface is implemented using React, and the transliteration functionality based on character mapping is created. The server side is built using Node.js. Examples of the functioning of the developed solution and the possibilities of its further development are given.

Keywords: transliteration, language layout, web application, React, Node.js, automatic text correction, browser extension

ЗМІСТ

ВСТУП

На даний момент у світі інформаційних технологій ефективність взаємодії з різного роду системами має велике значення для особистого користування та для професійної діяльності. Велика кількість щоденної комунікації, ділового листування, роботи з документами та іншими текстовими матеріалами здійснюється через клавіатуру. Та все ж навіть така повсякденна дія, як набір тексту, може відбуватися із помилками, зокрема якщо користувач під час введення тексту не поміняв розкладку.

Для користувачів, які активно працюють з текстом, часто виникає ситуація, коли текст вводиться на англійській розкладці клавіатури. Ці помилки сильно впливають на багато аспектів у роботі, що ускладнює сприйняття інформації та вимагає додаткових дій для виправлення. Такі помилки можуть спричинити втрату часу, зниження якості спілкування або навіть непорозуміння в діловому середовищі. У наш час, коли кожна секунда важлива, автоматизація виправлення таких ситуацій стає актуальною задачею.

За таких обставин є попит на створення програм, які б допомагали виправляти схожі помилки без необхідності втручання користувача. Саме таке завдання ставиться у моїй кваліфікаційній роботі — розробити розширення для браузера, що допомагає виправити текст введений українською мовою на англійській розкладці, і трансформує його у коректний український текст чи навпаки.

Під час реалізації було використано сучасний технологічний стек, що включає React для розробки клієнтської частини та Node.js для обробки логіки на сервері. React дозволив створити гнучкий та зручний інтерфейс, у якому користувачі можуть вводити текст, одразу бачачи результат перетворення. Node.js, зі свого боку, забезпечив ефективну реалізацію логіки транслітерації, в якій кожен символ англійської розкладки зіставляється з відповідним українським символом на основі попередньо визначених функцій.

У процесі дослідження було проаналізовано існуючі інструменти, які

частково або повністю вирішують аналогічні задачі. Проте більшість із них або вимагають активного перемикання режимів, або не підтримують українську мову. Це стало підставою для створення власного рішення, орієнтованого саме на потреби україномовних користувачів.

Розроблене рішення є максимально простим для інтеграції у робочий процес: воно не потребує встановлення складного програмного забезпечення, а у форматі розширення для браузера може працювати у будь-якому середовищі — як у текстових редакторах онлайн, так і у полях вводу на веб-сайтах. Завдяки цьому користувачі можуть уникати повторюваних помилок, не відволікаючись на ручне виправлення тексту.

Запропонований підхід також дозволяє розширити функціональність у майбутньому — наприклад, включити підтримку зворотного перетворення (з української на англійську розкладку), інтеграцію з сервісами автозаміни, перевірки правопису, або додати можливість миттєвого перекладу. Такі перспективи відкривають широкі можливості для розвитку продукту.

У цій роботі буде детально описано процес створення системи: від етапу формулювання проблеми до реалізації інтерфейсу та основної логіки. Окрему увагу приділено моделі транслітерації, методам тестування, аналізу ефективності рішення та можливостям подальшого вдосконалення.

Таким чином, основною метою даної кваліфікаційної роботи є створення інтуїтивно зрозумілого, надійного та ефективного інструменту, що покращує зручність набору українського тексту та мінімізує кількість механічних помилок, викликаних неправильним перемиканням мовної розкладки.

1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Огляд конкурентів

Не зважаючи на те що такі продукти не надто відомі у мережі та різних інтернет ресурсах, все-таки є цікаві інструменти для вирішеня таких проблем. Вони мають подібну мету: допомогти користувачеві заощадити час і зусилля, які могли б бути витрачені на ручне переписування тексту. Однак при більш детальному розгляді виявляється, що такі інструменти мають обмеження — як функціональні, так і технічні, що не дозволяють вважати їх ідеальними або універсальними рішеннями.

Першим прикладом є онлайн-сервіс за адресою https://tools.sochka.com/raskladka.html. Він представляє собою досить просту у використанні веб-сторінку, яка пропонує основну функцію — конвертацію тексту, надрукованого не тією мовною розкладкою, у відповідний текст українською мовою. Принцип дії сервісу полягає у зіставленні кожного введеного символу з його аналогом на альтернативній розкладці, що дає змогу швидко отримати коректний текст. Проте, при всій зручності, сервіс не є гнучким у використанні: користувач змушений самостійно скопіювати неправильний текст у спеціальне поле на сайті, обрати мову, натиснути кнопку обробки та скопіювати результат назад. Це перетворює процес на послідовність однотипних дій, що не вписуються в сучасне уявлення про автоматизацію.

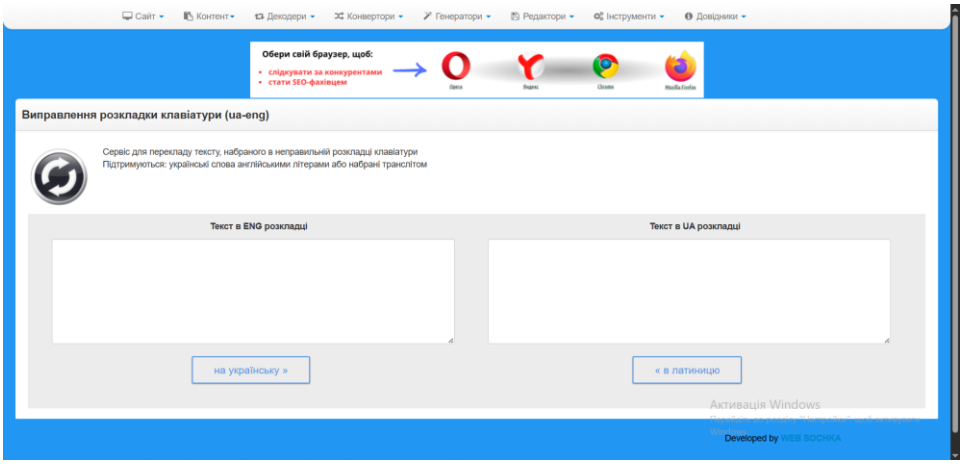

Рис. 1.1.1 – Вигляд сайту tools.sochka.com/raskladka.html [1].

Сайт повністю виконує свою функцію, хоч і рішення та вигляд сайт є застарілим. Відсутність браузерних розширень або вбудованої інтеграції в сторонні додатки суттєво обмежує сферу його використання. Сервіс не підтримує роботу у фоновому режимі, не реагує на текст у реальному часі та не запам'ятовує вподобання користувача. Хоча сама функція працює швидко, це все ж не той рівень зручності, якого очікує сучасний користувач, орієнтований на інтуїтивно зрозумілу та постійно доступну допомогу.

Другим прикладом є сайт https://design.te.ua/tools/rozkladka/, що має схожий функціонал. Сторінка також дозволяє вводити текст вручну у спеціальне поле та отримувати перетворений текст. Сервіс пропонує можливість перемикання між англійською та українською, а також має короткий опис того, для чого він може бути корисний. У плані дизайну він є трохи сучаснішим за попередній приклад, однак і тут відсутня будь-яка інтеграція з іншими платформами, браузерами чи вебзастосунками. Він, так само як і перший конкурент, існує виключно у форматі веб-сторінки, яка працює лише за прямого звернення користувача до неї.

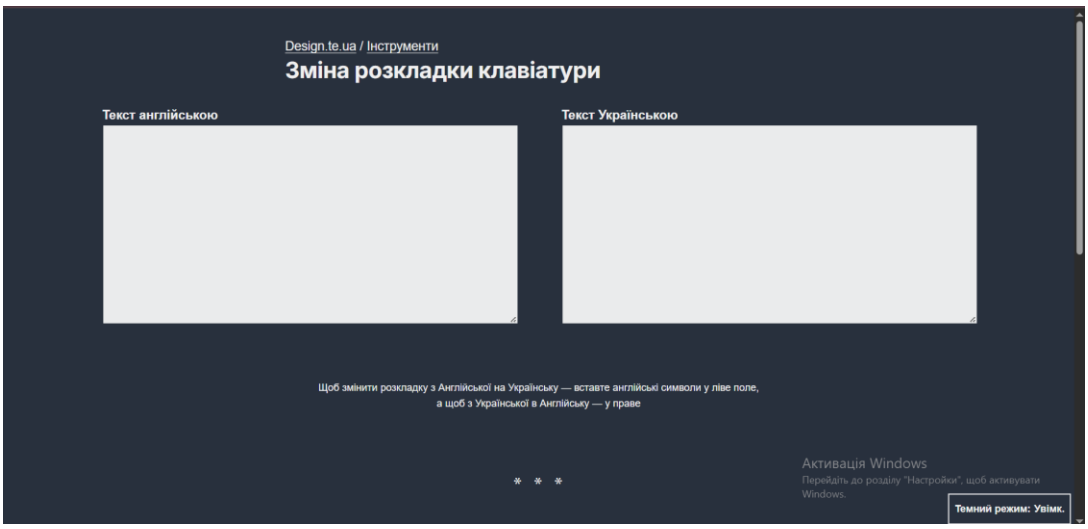

Рис. 1.1.2 – Вигляд сайту https://design.te.ua/tools/rozkladka/ [2].

Цей сайт на відмінну від попереднього має автоматичне перетворення тексту, це звільняє користувача від додаткових дій і забезпечує швидшу роботу із завданнями.

Обидва проаналізовані сервіси мають певну користь у вирішенні конкретного завдання, однак вони демонструють спільну рису: пасивність. Користувач змушений самостійно вирішувати, коли і як скористатися інструментом. У випадку, якщо він забуде про помилку, або ж не помітить її одразу, ніякої автоматичної допомоги не буде надано. Окрім того, обидва сервіси не передбачають збереження історії виправлень, що може бути критично важливим у деяких робочих сценаріях, наприклад — при редагуванні складних текстів.

Усе вищезгадане свідчить про наявність значного простору для вдосконалення та розвитку цього напрямку. В умовах зростання попиту на інструменти, що покращують взаємодію користувача з текстом, особливо українськомовного користувача, стає очевидною необхідність створення більш інтегрованого, зручного та адаптованого рішення. І саме ця ідея лягла в основу розробки власного проєкту — розширення для браузера, що автоматично виявляє та виправляє текст, введений у неправильній розкладці, без необхідності копіювання, перемикання між вкладками чи переходу на сторонні сайти.

1.2. Порівняння конкурентів

У попередньому розділі ми розглянули два онлайн-сервіси, що реалізують функцію виправлення тексту, набраного не тією мовною розкладкою клавіатури: це сторінки «Зміна розкладки» від Tools Sochka та сервіс на сайті Design.te.ua. Для того, щоб краще визначити переваги й недоліки існуючих рішень, а також зрозуміти, як вдосконалити власний продукт, варто провести детальне порівняння цих ресурсів. Це дозволить не лише уникнути повторення слабких місць конкурентів, а й взяти за основу їхні вдалі рішення, покращивши їх або розширивши можливості.

Сервіс Tools Sochka - це мінімалістичне та функціональне рішення. Перевагою даного сайту є простота за якої користувач вставляє текст у поле, натискає кнопку, і миттєво отримує виправлений варіант. Сторінка працює швидко, без перенавантаження візуальними елементами або зайвими функціями. Великим плюсом є мінімальна кількість реклами та будь-якої комерційної складової.

Ще однією перевагою є те, що сервіс працює без оновлення сторінки, а результат з'являється миттєво, що економить час користувача.

Однак така простота та мінімалістичність може бути й недоліком. Сервіс не має жодних можливостей персоналізації, не зберігає історії введених текстів, немає підтримки інших мовних пар, окрім стандартної «UA/EN». Крім того, відсутні індикатори або підказки для нових користувачів. Інтерфейс не масштабований для мобільних пристроїв — хоча працює, але не оптимізований. Також немає кнопки «копіювати результат» або «очистити», що ускладнює роботу під час масового використання.

Сайт design.te.ua/tools/rozkladka/ візуально виглядає дещо краще. Він має приємніший інтерфейс, а також включає в себе зручні кнопки для копіювання та інструкції щодо використання. Крім того, він орієнтований на дизайнерську спільноту, тому загальна візуальна структура виглядає привабливіше, що може

бути важливим для сучасного користувача. Інструмент також швидкий і працює без перезавантаження сторінки.

Однак сервіс від Design.te.ua також має обмеження. Він не надає можливості автоматичного визначення мови введення — користувач має самостійно здогадатися, коли саме потрібно виправити розкладку. Як і в першому прикладі, він підтримує лише українсько-англійську пару розкладок, і не дає можливості працювати з розширеними мовами. Також відсутня адаптація під роботу з великими текстами — все орієнтовано на швидке виправлення коротких фраз або слів.

У порівнянні можна скласти висновок, що обидва сервіси виконують своє основне завдання, але мають вузьку функціональність і обмежений набір інструментів. Їхній підхід направлений на мінімалістичність та простоту є ефективним, однак не охоплює ширшої аудиторії, яка могла б потребувати додаткових можливостей: розширеної мовної підтримки, інтеграції з браузерами або мобільними додатками, збереження історії введень, персоналізації інтерфейсу або навіть автоматичного розпізнавання неправильного тексту.

Таким чином, обидва конкуренти є хорошими прикладами функціональності, однак мій проєкт має на меті поєднати їхні найкращі сторони й водночас розширити потенціал для більш вимогливих користувачів. Саме такий підхід дозволить створити дійсно конкурентоспроможний продукт на тлі вже існуючих інструментів.

1.3. Обґрунтування вибору напрямку дослідження

У сучасному інформаційному суспільстві обсяг текстової інформації, з якою взаємодіє користувач у повсякденному житті, стрімко зростає. Комунікація в месенджерах, створення повідомлень у соціальних мережах, ділове листування, заповнення форм, публікація контенту на платформах типу Google Docs, Notion

та ріних соціальних мережах — усе це передбачає активне використання клавіатури. Часто трапляються ситуації, коли текст вводиться не тією мовною розкладкою, особливо в контексті перемикання між англійською та українською мовами. Виправлення подібних помилок вручну є трудомістким і відволікає від основної задачі.

Веб-розробка включає створення та підтримку веб-сайтів і веб-додатків за допомогою різних мов програмування, таких як Java, Python та JavaScript. Розробники використовують ці мови для написання програмного забезпечення, яке забезпечує функціональність та інтерактивність веб-сайту, роблячи його доступним як на настільних комп'ютерах, так і на мобільних пристроях. Веб-розробка охоплює все: від кодування та управління базами даних до дизайну інтерфейсу користувача, забезпечуючи безперебійний онлайн-досвід для користувачів. Наведені нижче навчальні посібники містять поради та рекомендації щодо того, як розпочати роботу з веб-розробкою.

Для цієї проблеми розробка розширення для браузера, яке буде допомагати у таких ситуація, може стати чудовим рішенням. Таких підхід звільнить він потрібності завантажувати програми на власний комп1ютер а також дасть ефективну функціональність та швидкість у наборі текссту. І на мою думку саме це рішення буде зручним для усіх, через те що зараз переважна більшість користувачів пише тексти на сайтах та програмах які працюють у браузері.

Розширення Google Chrome — це програмні надбудови, які розширюють функціональні можливості браузера Google Chrome. Вони створюються із застосуванням мов JavaScript, HTML та CSS. Розробникам надається доступ до Chrome API, за допомогою якого можна взаємодіяти з вкладками, закладками, керувати процесом завантажень і переглядати історію відвідувань. Також доступні опції для зміни параметрів браузера, взаємодії з іншими розширеннями та виконання багатьох інших дій [3].

Доступні інструменти,які було знайдено:

- tools.sochka.com/raskladka.html,
- design.te.ua/tools/rozkladka.

Вони пропонують базовий функціонал виправлення тексту, проте їх використання вимагає копіювання тексту, переходу на окрему сторінку, натискання кнопки та повторного копіювання результату. Це не лише забирає зайвий час, а й створює додаткове когнітивне навантаження, що суперечить принципам user-friendly інтерфейсів.

У зв’язку з цим, актуальним стало завдання створити браузерне розширення, яке б інтегрувалося безпосередньо у веб-середовище користувача та забезпечувало автоматичне або напівручне виправлення тексту в будь-якому місці введення. Такий підхід дозволяє працювати швидко, інтуїтивно, не покидаючи активного вікна браузера. Крім того, розширення не потребує встановлення додаткового програмного забезпечення на комп’ютер, а значить, користувачі швидше адаптуються до роботи із такою системою.

Для створення інтерфейсу клієнтської частини було використано бібліотеку React, оскільки вона відзначається високою продуктивністю, гнучкістю та простотою в обслуговуванні. React забезпечує можливість створення багаторазово використовуваних компонентів, що полегшує організацію інтерфейсу та прискорює процес розробки. Крім того, бібліотека активно підтримується Meta (Facebook), що забезпечує постійне оновлення, стабільність та широку спільноту розробників. Порівняно з іншими популярними фреймворками, такими як Angular та Vue, React має нижчий поріг входу, кращу адаптованість до проєктів різного масштабу та широку екосистему бібліотек для стану (наприклад, Redux, Zustand) і маршрутизації (React Router). Angular є повноцінним фреймворком з більшою складністю, жорсткою структурою та потребою в TypeScript, що ускладнює його використання для невеликих чгнучких проєктів. Vue, хоча і вважається простим для початку, менш поширений в комерційних проєктах і має слабшу інтеграцію з корпоративними інструментами. Вибір React забезпечує баланс між легкістю, масштабованістю та підтримкою, що робить його ідеальним варіантом для створення браузерних розширень.

React застосовується для створення інтерфейсу користувача у веб-додатках. Головне призначення React Frontend полягає у розробці динамічних,

інтерактивних та чутливих до дій користувача інтерфейсів. Використання React на фронтенді дає змогу будувати функціонально насичені та інтерактивні додатки з високою швидкістю рендерингу та плавним переходом між сторінками [4].

React є бібліотекою, яка дозволяє комбінувати інтерфейсні компоненти, але не задає чітких рішень для маршрутизації або отримання даних. Для створення повноцінного застосунку рекомендується використовувати фреймворк повного стеку на базі React, наприклад Next.js або Remix [5].

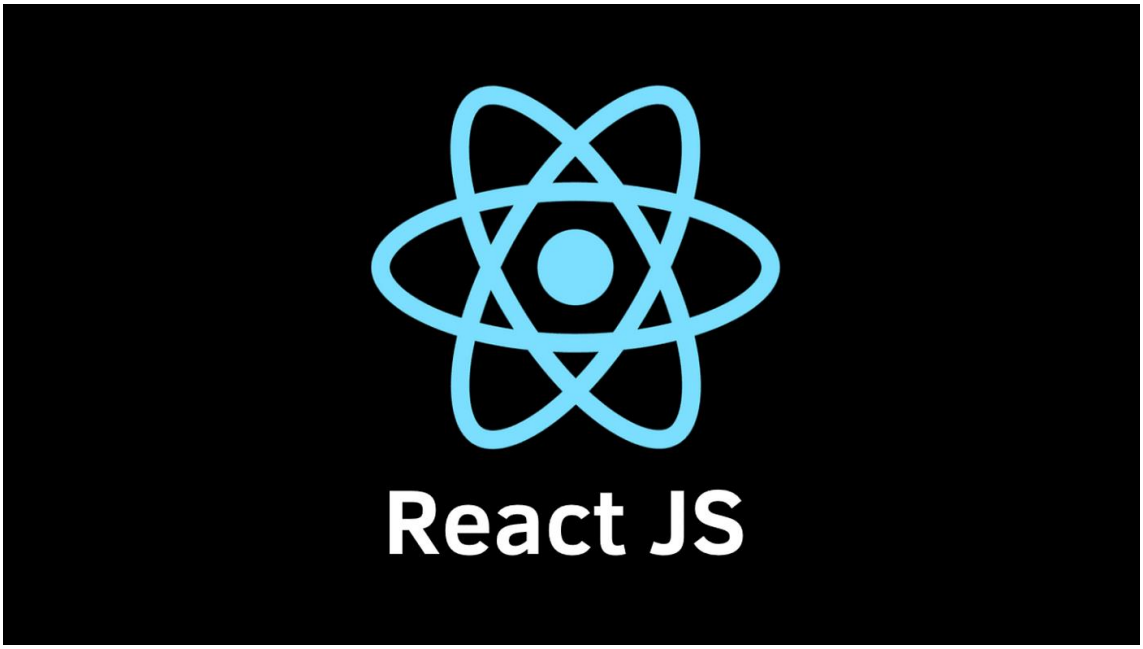

Рис. 1.3.1 - Логотип бібліотеки React JS

Node.js - це високопродуктивна платформа, яка за останні роки докорінно змінила підхід до створення веб-застосунків. Її головна концепція полягає в тому, щоб перетворити вже поширену мову JavaScript на універсальний інструмент, придатний як для клієнтської, так і для серверної частини розробки.

Ключовою особливістю Node.js є використання асинхронної моделі програмування. Це дає змогу розробникам створювати високопродуктивні та чутливі до користувача застосунки, здатні обробляти велику кількість одночасних запитів. Платформа також вирізняється малою вагою та високою швидкодією, що сприяє ефективному використанню серверних ресурсів.

Node.js здобув велику популярність як основа для створення веб-серверів,

API, мікросервісів та інших рішень. У цьому матеріалі буде розглянуто переваги й недоліки платформи, її основні можливості, архітектуру та підходи до розробки застосунків. Також буде наведено приклади використання Node.js для кращого розуміння її практичного застосування у створенні сучасних серверних систем [6].

Додатки на Node.js виконуються в межах одного процесу, не створюючи окремих потоків для кожного запиту. У стандартній бібліотеці Node.js реалізовано набір асинхронних інструментів вводу/виводу, які запобігають блокуванню JavaScript-коду. Більшість бібліотек у Node.js дотримуються неблокуючого підходу, що робить блокуючі дії радше винятком, ніж правилом. Під час виконання операцій вводу/виводу - як-от зчитування з мережі, звернення до бази даних чи файлової системи - Node.js не блокує потік і не витрачає ресурси процесора на очікування. Замість цього виконання продовжується після надходження відповіді.

Такий підхід дозволяє Node.js обробляти тисячі одночасних підключень на одному сервері без необхідності керувати багатьма потоками, що часто є джерелом помилок у традиційних багатопоточних моделях [7].

Рішенням для реалізації браузерного розширення стала мова JavaScript. Оськільки вона є широко використовується в веб-розробці, особливо в контексті створення інтерфейсів користувача та взаємодії з DOM-елементами. JavaScript забезпечує безпосередній доступ до браузерного API, включаючи роботу з розширеннями для Chrome, Edge чи Firefox, що дозволяє реалізовувати динамічну поведінку елементів сторінки без потреби в компіляції чи сторонньому програмному забезпеченні. Гнучкість та універсальність мови роблять її оптимальним вибором для завдань, пов’язаних із модифікацією тексту в режимі реального часу, інтеграцією з формами вводу та забезпеченням високої інтерактивності.

JavaScript - це мова програмування високого рівня, яка підтримує імперативну, функціональну та подієво-орієнтовану парадигми. Вона має динамічну типізацію та використовується для створення послідовностей дій - «скриптів» або «сценаріїв». Такі скрипти, як правило, інтерпретуються під час

виконання, а не компілюються заздалегідь, тому не вимагають додаткових інструментів для перетворення у машинний код.

Кожен сучасний веб-сайт або веб-додаток створюється за допомогою трьох основних технологій — HTML, CSS і JavaScript. Саме JavaScript є основним інструментом для реалізації інтерактивної логіки та взаємодії з користувачем.

Сьогодні дедалі більше компаній використовують JavaScript не лише на стороні клієнта, а й на сервері — за допомогою платформи Node.js, яка забезпечує середовище виконання JavaScript-коду для серверних застосунків [8].

1.4. Вибір методології розробки

Одним із важливих етапів у процесі створення програмного забезпечення є вибір методології розробки. Це впорядкований план дій, що дозволяє розуміти очікуваний результат, способи його досягнення та інструменти, які варто застосовувати. Методологія створення програмного забезпечення - це сукупність випробуваних підходів і практик, які забезпечують якісну та ефективну реалізацію цифрових продуктів. В ІТ-галузі сформувалося кілька основних моделей розробки ПЗ, на які варто звернути увагу.

Кожен проєкт має свій життєвий цикл - послідовність етапів, через які проходить розробка. Все починається зі створення ідеї та ухвалення рішення: наприклад, компанія вирішує створити сайт або мобільний застосунок — і на цьому моменті вже запускається життєвий цикл.

Наступним кроком є етап підготовки й аналітики, де формулюється концепція та визначається напрям створення продукту. Коли сформоване чітке уявлення про кінцевий результат, слід обрати шляхи його досягнення. На аналітичному етапі ідея трансформується в покроковий план, визначається стек технологій, а також здійснюється вибір методології розробки.

Це означає обрання конкретного підходу до створення продукту. Саме від

методики залежить, наскільки якісним буде кінцевий результат. Вибрана модель визначає порядок реалізації завдань, формує систему контролю, оцінки прогресу, а також дозволяє спланувати терміни виконання та вартість. Методологія розробки забезпечує стабільність у процесі, а це є однією з головних цілей проєкту [9].

Однією з найстаріших і найбільш структурованих методологій є методологія Waterafall. Її головна особливість полягає в послідовному виконанні етапів проєкту: від збору вимог до проєктування, потім розробки, тестування, впровадження та, нарешті, підтримки. Кожен етап передбачає повне завершення попереднього, після чого можливість повернення назад зазвичай відсутня. Така модель добре підходить для проєктів, де вимоги точно відомі заздалегідь і не змінюються в процесі розробки, наприклад, у разі створення програм для держустанов чи промислових систем. Проте її жорстка структура часто стає недоліком, особливо в динамічних середовищах, де вимоги змінюються або виникає потреба в частому тестуванні нових функцій. Виявлення помилок на пізніх етапах може виявитися критичним, а адаптація під нові умови — практично неможливою без повного перероблення.

Методологія Waterfall – це підхід до управління проектами, який наголошує на лінійному прогресі від початку до кінця проекту. Ця методологія, яку часто використовують інженери, орієнтована на попереднє планування, детальну документацію та послідовне виконання [10].

Зі зростанням потреб у більш гнучкому підході до розробки з’явилися методології, засновані на принципах Agile. Однією з них є Kanban - візуальна система управління процесами, яка передбачає постійний потік задач із мінімальними обмеженнями. Основним інструментом тут є канбан-дошка, на якій задачі переміщуються між колонками відповідно до стадій виконання. Методологія дозволяє команді працювати без чітких ітерацій чи часових меж, зосереджуючи увагу на ефективності та плавності виконання. Kanban часто використовується для підтримки та покращення вже існуючих продуктів, а також у тих випадках, коли є потреба швидко реагувати на зовнішні запити без

жорстких рамок спринтів. Проте така свобода іноді призводить до відсутності контролю над термінами або до скупчення незавершених завдань, якщо не дотримуватися дисципліни.

Третьою методологією, яка поєднує гнучкість Agile та структурованість процесу, є Scrum. Цей підхід став особливо популярним у розробці веб-застосунків і стартап-проєктів. Scrum передбачає розбиття роботи на короткі часові інтервали - спринти, тривалістю від одного до чотирьох тижнів. На початку кожного спринту команда спільно з власником продукту визначає перелік завдань, які потрібно реалізувати. У процесі роботи щодня проводяться короткі мітинги (daily stand-ups), де кожен учасник доповідає про зроблене, плани та можливі перешкоди. Завершенням кожного спринту є демонстрація результату та ретроспектива, на якій аналізуються успіхи й недоліки спільної роботи [11].

Однією з найбільших переваг Scrum є можливість швидко адаптуватися до змін у вимогах, що особливо важливо в умовах, коли проєкт постійно розвивається або залежить від зворотного зв’язку користувачів. Команда при цьому самостійно організовує свою роботу, а роль Scrum-майстра полягає в усуненні зовнішніх бар’єрів і підтримці динаміки процесу. Такий підхід вимагає від учасників високого рівня самодисципліни та вміння працювати у команді. Хоча іноді Scrum здається складним у впровадженні для початківців, саме він дає змогу побудувати гнучку систему постійного вдосконалення продукту.

У межах цього проєкту було вирішено використати саме Scrum як основну методологію. Причин для цього кілька. По-перше, проєкт є динамічним, і його функціонал може змінюватися в процесі реалізації відповідно до результатів тестування та зворотного зв’язку. По-друге, Scrum дозволяє реалізовувати продукт поетапно, поступово додаючи нові можливості, що ідеально підходить для індивідуального або невеликого командного проєкту. По-третє, можливість постійного контролю за прогресом, оцінка реального обсягу роботи та впровадження змін після кожного спринту робить Scrum універсальним інструментом, особливо у випадку обмежених ресурсів.

1.5. Формування вимог до системи

Формування вимог є одним із ключових етапі у розробці проекту. На основі попередніх розділів, зокрема опису конкурентів, ми можемо скласти певний стек того що може мати програма. Коректно сформульовані вимоги дозволяють уникнути непорозумінь на етапі реалізації, мінімізувати ризики, зменшити витрати часу та ресурсів, а також забезпечити відповідність розробленої системи поставленим цілям.

Функціональні вимоги

Розширення повинно містити такі функціональні вимоги:

- Редагування тексту
- Копіювання тексту
- Увімкнення/вимкнення
- Чорний список сайтів
- Автоматичне виявлення помилок
- Застосування виправлень

Крім того, сайт має відповідати певному переліку нефункціональних вимог. Важливими нефункціональними вимогами є:

Нефункціональні вимоги.

- Продуктивність: швидкість обробки: час перевірки тексту не повинен перевищувати 2 секунди для тексту середнього розміру, також споживання ресурсів та відгук інтерфейсу повинен бути мінімальним.
- Зручність: розширення має мати інтуїтивний дизайн, мінімалістичні стилі та зручний спосіб розміщення функцій.
- Кросплатформенність: Підтримка різних браузерів(Chrome, Edge).
- Безпека та приватність: перевірка тексту повинна відбуватися локально та не перенаправлятися.
- Стабільність: розширення не повинно викликати зависання або збої браузера

1.6. Проєктування відношень між акторами та прецедентами

Розширення для браузера передбачає тільки одного актора:

- Користувач:
  - Має доступ до веб-браузера з встановленим розширенням
  - Використовує різні мови для написання тексту
  - Працює з різними типами веб-сайтів та текстових полів

Наведений приклад демонструє зв’язок користувача із розширенням. Зробимо таблицю для структуризації даних.

Таблиця 1.6.1 – Варіанти використання системи

Код	Основний актор	Найменування	Опис
К1	Користувач	Вхід на сторінку магазину	Користувач може увійти на сайт магазину розширень браузера
К2	Користувач	Пошук розширення	Користувач шукає розширення для перевірки текстових помилок
К3	Користувач	Встановлення	Користувач може встановити розширення
К4	Користувач	Налаштування	Користувач може налаштувати мову для перевірки
К5	Користувач	Локальне користування	Користувач встановлює розширення з локального файлу (для розробників)
К6	Користувач	Видалення розширення	Користувач може видалити розширення із свого браузера чи на сторінці магазину розширень

Проведений аналіз варіантів використання демонструє, що система браузерного розширення має чітко структурований функціонал, хоч і взаємодія відбувається з одним актором - користувачем браузера. Виділені шість основних

прецедентів охоплюють повний життєвий цикл роботи з розширенням: від початкового встановлення та налаштування до активного використання функцій перевірки і виправлення тексту.

Щоб наочно показати взаємозв'язки між актором і прецедентами на основі проведеного аналізу, доцільно створити діаграму варіантів використання. Діаграма варіантів використання в UML є ключовим інструментом для формулювання вимог до нової системи браузерного розширення. Вона відображає передбачувану поведінку системи автоматичного виправлення текстових помилок, при цьому не деталізуючи конкретні способи реалізації за допомогою NodeJS та React.

Головна концепція такого моделювання полягає в проєктуванні системи розширення з позиції кінцевого користувача браузера. Це дієвий підхід для опису поведінки системи зрозумілою мовою, що демонструє всю зовнішню функціональність розширення, спрямованого на автоматичне виправлення помилок у тексті. Діаграма слугує наочним способом відображення взаємодії користувача з ключовими можливостями системи: перевіркою тексту, виправленням помилок, керуванням параметрами та аналізом статистичних даних.

Рис. 1.6.1 - Діаграма варіантів використання

Тож у результаті проведеного аналізу предметної області розширення для виправлення текстових помилок було детально досліджено всі ключові аспекти майбутньої системи. Сформовані функціональні та нефункціональні вимоги створюють комплексну основу для технічної реалізації розширення з використанням NodeJS та React.

Проєктування відношень між акторами та прецедентами виявило, що система орієнтована на роботу з єдиним актором - користувачем браузера, для якого визначено шість основних варіантів використання. Ці прецеденти охоплюють повний спектр взаємодії користувача з розширенням. Виявлені взаємозв'язки між прецедентами (включення, розширення та узагальнення) демонструють логічну структурованість системи.

2. РОЗРОБКА МОДЕЛІ ТА ПРОГРАМНОГО КОМПЛЕКСУ

2.1. Архітектурна модель

Однією із важливих рішень є вибір архітектури. Правильні архітектурні рішення на початкових етапах розробки визначають не лише технічну якість кінцевого продукту, але й його здатність до адаптації та подальшого розвитку. Для системи автоматизованого виявлення та корекції текстових неточностей архітектурний дизайн стає особливо критичним, оскільки розширення повинно забезпечувати швидку обробку контенту без порушення користувацького досвіду. Архітектурне рішення для браузерного розширення автоматизованої корекції тексту грунтується на принципах компонентно-орієнтованого проектування з акцентом на розподіленій відповідальності модулів. Основою архітектурної стратегії є багаторівнева організація системи, що гарантує ефективну комунікацію між елементами та строге дотримання безпекових обмежень браузерного оточення.

Архітектура програмного забезпечення (ПЗ) — це структурний план, який визначає, як організована система, яким чином її окремі компоненти взаємодіють між собою та з іншими зовнішніми елементами. При розробці архітектури застосовуються спеціальні шаблони, що дають змогу використовувати найефективніші рішення для вирішення типових архітектурних завдань. Її можна порівняти з планом будівлі: є кімнати, двері, стіни та вікна — тільки замість фізичних матеріалів використовуються код, сервери й бази даних. Архітектура має ключове значення, оскільки саме вона визначає, наскільки система буде стабільною, масштабованою, безпечною та зручною для подальшого вдосконалення [12].

При створенні сучасних веб-застосунків, зокрема браузерних розширень, критично важливою є гнучкість архітектури, зручність підтримки коду та можливість масштабування. Саме тому для реалізації даної системи було обрано компонентно-орієнтований підхід, який є одним із найвикористовуваніших у

сучасному фронтенд-розробленні, зокрема в екосистемі React.

Компонентно-орієнтована модель передбачає поділ інтерфейсу на окремі незалежні частини - компоненти, кожен з яких відповідає за власну логіку, стан і відображення. У випадку даного розширення основні функції - введення тексту, відображення результату, копіювання, перемикання мови транслітерації - реалізовані у вигляді окремих компонентів або логічно ізольованих блоків у межах одного компонента. Такий підхід дозволив досягти більшої модульності та повторного використання коду.

У порівнянні з класичними підходами, де вся логіка інтерфейсу реалізується у вигляді єдиного монолітного коду, компонентна архітектура значно полегшує підтримку, дає змогу командній розробці та дозволяє зосередитися на вузьких завданнях без необхідності заглиблюватися в усю систему. Це особливо важливо у проєктах, які можуть у майбутньому розвиватися або змінюватися.

2.1.1. Розробка моделі предметної області

Створення моделі предметної області є фундаментальним етапом проектування системи автоматизованої корекції текстів, оскільки визначає концептуальний каркас всіх бізнес-сутностей та їх взаємозв'язків. Модель предметної області служить мостом між вимогами користувачів та технічною реалізацією, забезпечуючи чітке розуміння контексту роботи системи всіма учасниками проекту.

Враховуючи сучасні тенденції у розробці веб-застосунків, зокрема зростання популярності односторінкових додатків (SPA), зростання вимог до продуктивності, а також зручності для кінцевого користувача, особливу увагу слід приділити створенню моделі предметної області. Саме з неї починається процес системної розробки — вона визначає ключові сутності, їхні властивості, зв’язки між об'єктами, бізнес-логіку та сценарії взаємодії користувача з системою. Це

дозволяє сформувати цілісне уявлення про функціональність, яку має реалізовувати програмний продукт, та закласти структуровану основу для подальшої реалізації.

SPA (Single Page Application, або односторінковий застосунок) — це підхід до розробки веб-додатків, за якого завантажується лише одна HTML-сторінка, а подальше оновлення контенту відбувається без повного перезавантаження сторінки в браузері. Основу роботи таких застосунків становлять API-інтерфейси JavaScript, як-от XMLHttpRequest та Fetch, які відповідають за отримання нових даних із сервера у фоновому режимі. На сьогодні розробникам доступна велика кількість frontend-бібліотек і фреймворків, що значно спрощують створення SPA [13].

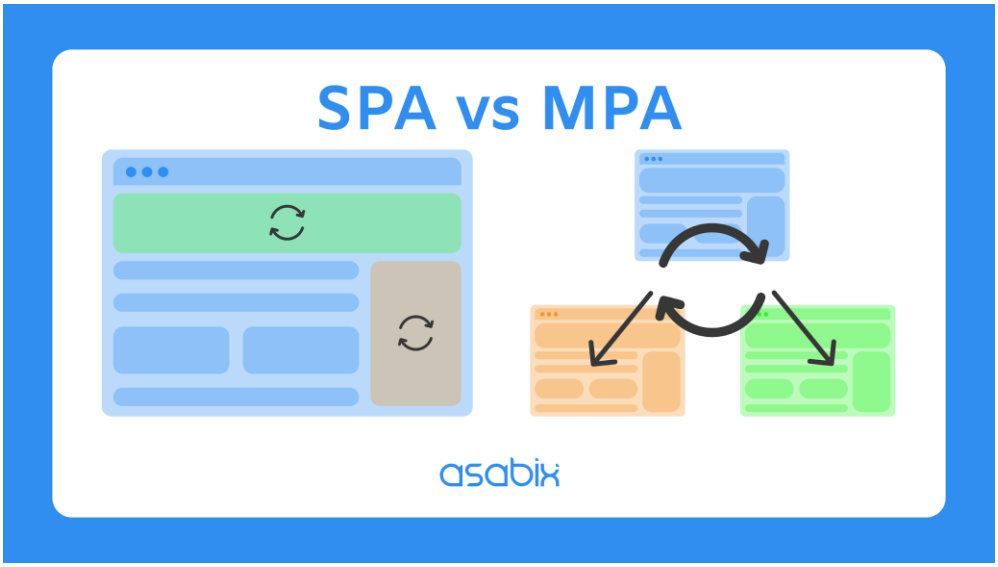

Рис. 2.1.1.1 - Демонстрація різниці між SPA та MPA [14].

Модель предметної області створюється з урахуванням як технічних, так і користувацьких вимог до майбутньої розробки. У нашому випадку йдеться про розширення для браузера, яке автоматично виправляє текстові помилки, пов’язані з введенням українських слів у невірній розкладці. Основна ідея полягає в тому, щоб користувач міг без додаткових зусиль вводити текст у будь-якому полі вводу, і у разі помилки система оперативно пропонувала або автоматично замінювала неправильно введене слово на правильний український варіант.

Також варто зазначити, що модель предметної області формує основу для перевірки відповідності системи вимогам. У майбутньому вона може бути використана для тестування, модульної перевірки окремих частин системи або ж для створення технічної документації. Враховуючи особливості предметної області, а також вимоги до швидкодії, інтерактивності та зручності користування, було прийнято рішення про використання сучасного архітектурного підходу — Single Page Application (SPA). Його реалізація здійснюється за допомогою бібліотеки React, що дозволяє створювати гнучкий та динамічний інтерфейс користувача з можливістю миттєвого оновлення контенту без повного перезавантаження сторінки. У поєднанні з Vite, як сучасним інструментом для збирання й розгортання застосунку, це дозволяє досягти високої продуктивності на етапі розробки та під час виконання застосунку.

Крім того, використання React як бібліотеки для побудови клієнтської частини сприяє модульному поділу функціональності, що ідеально відповідає об'єктно-орієнтованій моделі, сформованій у рамках предметної області. Компонентний підхід React дозволяє чітко структурувати логіку взаємодії між основними сутностями системи (користувач, розширення, перевірка тексту тощо), а також забезпечити повторне використання коду й полегшити майбутнє тестування.

2.1.2. Проектування діаграми класів, опис основних класів, методів, компонентів.

У процесі проєктування архітектури важливо визначити загальну структуру програмного забезпечення, що забезпечить ефективну реалізацію функціональних вимог, визначених на етапі моделювання предметної області. Архітектура системи відіграє ключову роль у забезпеченні масштабованості, зручності супроводу та модульності застосунку. З огляду на вибір парадигми Single Page

Application, архітектура програмного забезпечення побудована на основі компонентного підходу. Це означає, що кожна частина функціональності реалізована у вигляді окремого компонента, який відповідає за певний фрагмент інтерфейсу або логіки. Компоненти взаємодіють між собою через чітко визначені інтерфейси, що значно спрощує їх тестування та повторне використання.

Бібліотека React, яка лежить в основі клієнтської частини застосунку, дозволяє реалізувати цю архітектуру завдяки гнучкій системі створення та рендерингу компонентів. Платформа Vite використовується для швидкої розробки та збірки, що прискорює процес інтеграції окремих частин системи. Для збереження даних застосовується локальне сховище браузера, яке взаємодіє з функціональністю перевірки введеного тексту у реальному часі.

У межах розробки браузерного розширення для виправлення текстових помилок клас User виконує роль логічної моделі користувача, який взаємодіє з розширенням у локальному середовищі браузера. Оскільки система наразі не передбачає автентифікації чи створення облікових записів, модель User реалізується в контексті збереження персональних налаштувань та поточного стану взаємодії з розширенням.

Основні поля класу:

- languagePreferences — список мов, обраних користувачем для автоматичної перевірки тексту. Визначає правила транслітерації або виправлення відповідно до вибраної мови.
- settings — об’єкт, що містить конфігураційні параметри розширення (наприклад, стиль виправлення, активація/деактивація вбудованих функцій, тема інтерфейсу).
- usageStats — статистика використання (опційно): історія виправленнь.

Така модель User не є повноцінним об’єктом бази даних, натомість реалізується через локальне збереження (LocalStorage або IndexedDB). Вона дозволяє створити відчуття персоналізованої взаємодії навіть без системи реєстрації.

У перспективі, структура класу User може бути розширена для підтримки

синхронізації між пристроями, використання хмарних профілів або інтеграції з обліковими записами браузера.

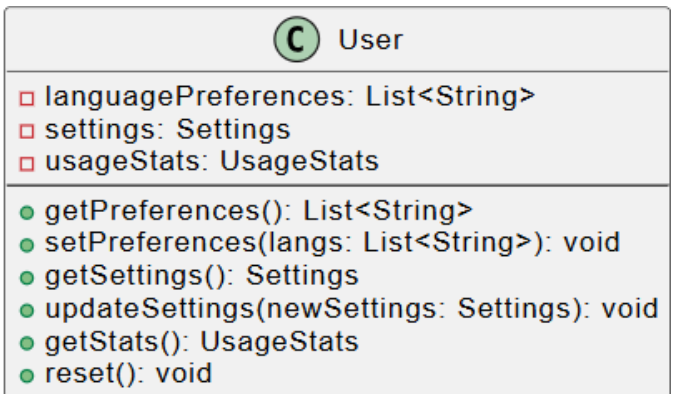

Рис. 2.1.2.1 - Клас User.

Клас Settings відповідає за збереження параметрів налаштувань користувача розширення. Він містить поля, що зберігають інформацію про вибрану мову перевірки тексту, активацію автоматичного виправлення, та інші параметри, які впливають на роботу розширення.

Основні поля класу:

- language - рядок, що визначає мову перевірки тексту (наприклад, "uk", "en").
- autoCorrect - булеве поле, що вказує, чи увімкнено автоматичне виправлення помилок.
- enabled - булеве поле, що визначає, чи активне розширення.
- customRules — список або колекція правил, які користувач може додати для більш індивідуальної перевірки.

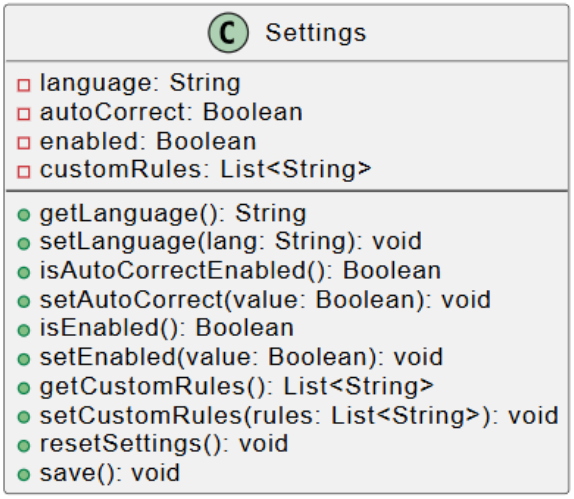

Рис. 2.1.2.2 - Клас Settings.

Клас TextCheckTask відповідає за одне конкретне завдання перевірки або виправлення тексту, яке ініціює користувач. Він містить основні дані, пов’язані з текстом для аналізу, статусом обробки і результатами.

Основні поля класу:

- inputText (string) — текст для перевірки.
- language (string) — вибрана мова для перевірки.
- correctedText (string) — текст після виправлення.
- settings (UserSettings) — об’єкт налаштувань, які застосовуються до цієї перевірки.

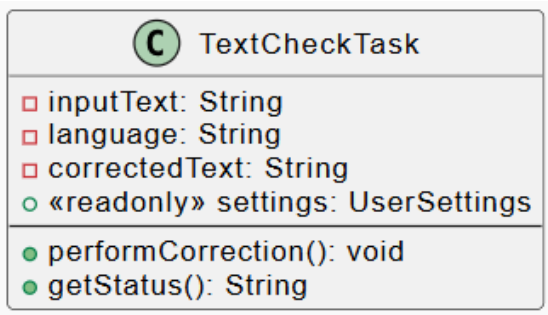

Рис. 2.1.2.3 - Клас TextCheckTask.

Клас Dictionary відповідає за збереження списку поширених помилок і їх правильних варіантів. Він використовується для автоматичного виправлення тексту в межах розширення.

Основні поля класу:

- entries(Map<string, string>) - містить символи різних мов для перевірки.

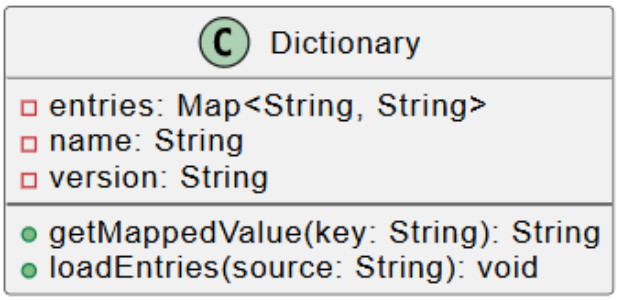

Рис. 2.1.2.4 - Клас Dictionary

Розроблена система автоматичного виправлення помилок у тексті побудована на модульній архітектурі, де кожен ключовий компонент представлений окремим класом зі строго визначеними обов'язками.

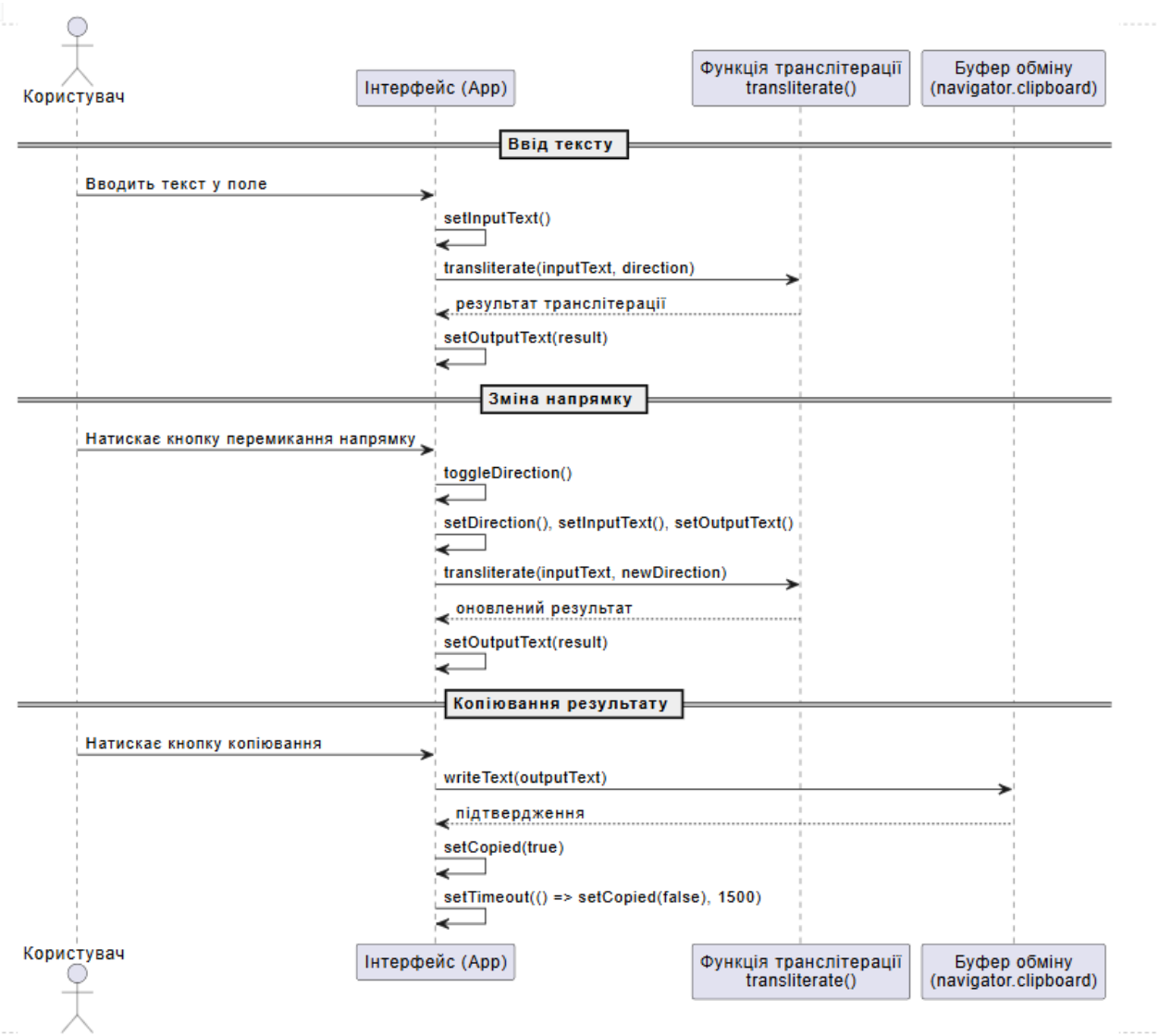

Рис. 2.1.2.5 - Діаграма послідовностей

На діаграмі послідовностей показано взаємодію користувача з інтерфейсом розширення під час трьох основних дій: введення тексту, перемикання напрямку транслітерації та копіювання результату. Вона відображає послідовність викликів функцій між компонентом App, функцією transliterate та буфером обміну, демонструючи, як кожна дія користувача спричиняє обробку даних і оновлення інтерфейсу у реальному часі.

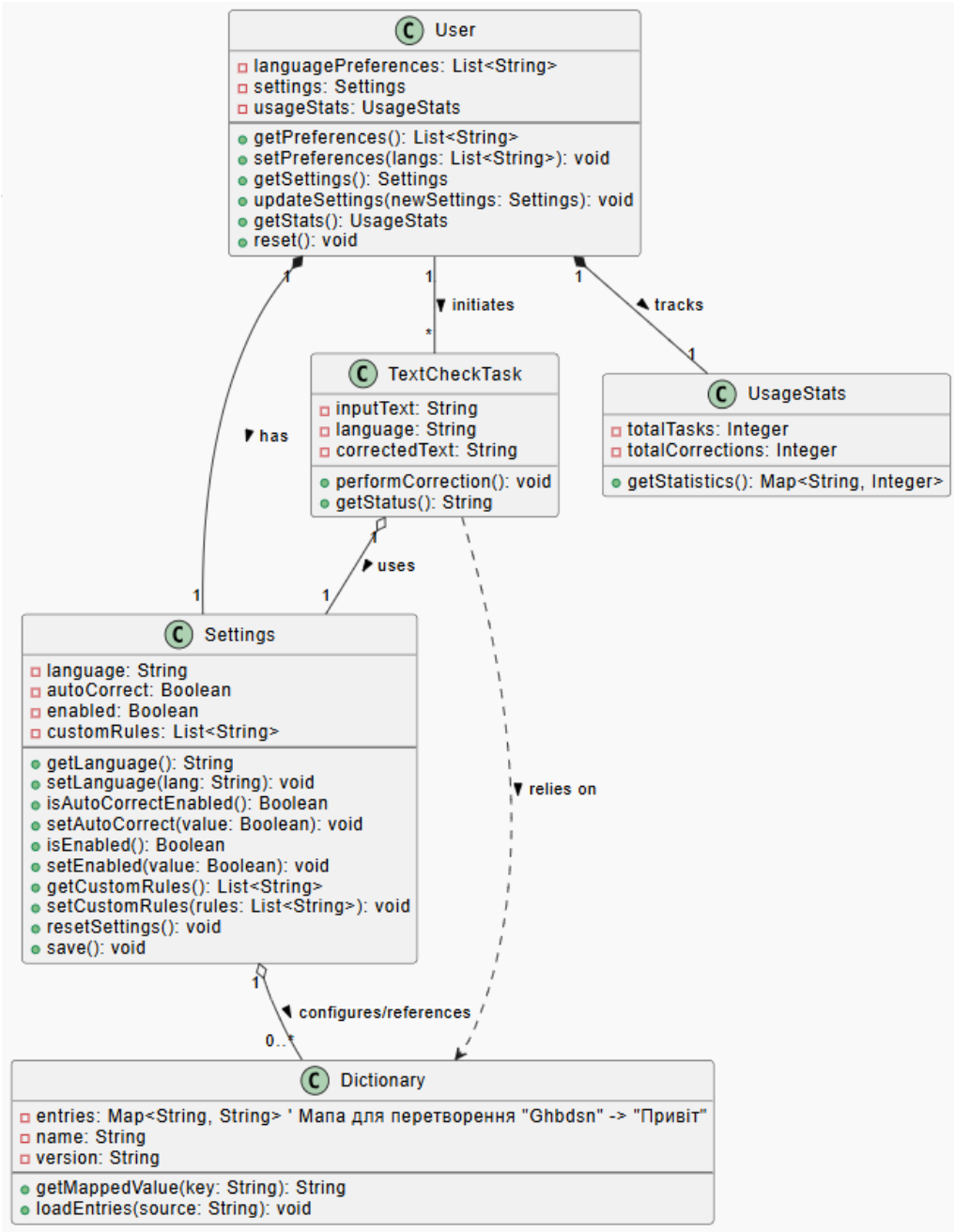

Рис. 2.1.2.6 - Діаграма зв’язків між описаними класами.

Побудована діаграма дозволила не лише ідентифікувати основні відповідальності кожного класу, але й визначити типи зв'язків (композиція, агрегація, асоціація, залежність), що є критично важливим для розуміння логіки взаємодії об'єктів та забезпечення цілісності даних. Це моделювання слугує фундаментальним архітектурним планом для подальшої ефективної реалізації та розробки програмного забезпечення, забезпечуючи структурований підхід до кодування та майбутньої масштабованості системи.

2.2. Конструювання веб-застосунку

Після завершення етапів моделювання та проєктування було розпочато безпосередню розробку програмного забезпечення. Основним об'єктом розробки є клієнтська частина - браузерне розширення, що працює локально. Такий підхід забезпечує швидку обробку даних, гарантує більшу конфіденційність введеної інформації та спрощує архітектуру загального рішення.

Розширення реалізоване за допомогою мови програмування JavaScript із використанням бібліотеки React для побудови динамічного користувацького інтерфейсу у вигляді SPA (Single Page Application). Для ефективного збирання та організації застосунку було використано Vite - сучасний інструмент для фронтенд-розробки, який дозволяє швидко запускати локальні версії, підтримує.

Розробка велася відповідно до специфікації Manifest v3, яка є сучасним стандартом для створення безпечних і продуктивних розширень у браузерах на базі Chromium. У цьому розділі буде розглянуто реалізацію ключових функціональних частин: логіки транслітерації, інтерфейсу користувача (вікно popup), збереження налаштувань користувача через chrome.storage, взаємодії з DOM-елементами сторінки, а також структури директорій та файлів проєкту [15].

У процесі конструювання системи важливим етапом стала організація правильного розгортання веб-застосунку, який було реалізовано як односторінковий інтерфейс користувача на базі React. Для забезпечення швидкої збірки, зручної роботи з модулями та сучасного підходу до розробки, було обрано інструмент Vite. Цей сучасний збирач забезпечує майже миттєве оновлення при збереженні змін у коді, а також оптимізує результуючий JavaScript і CSS для подальшого використання в браузері.

Vite дозволив створити високопродуктивне середовище для локальної розробки, де можливо зосередитися безпосередньо на розширенні функціоналу, не витрачаючи час на довгі цикли компіляції. У режимі розробки застосунок доступний за локальним хостом, що полегшує його налагодження й тестування.

Після розробки інтерфейсу і його збирання у вигляді статичних файлів, ці результати були використані для створення браузерного розширення. Щоб протестувати та налагодити його без публікації в офіційному магазині, було використано режим розробника (developer mode) у браузері. У цьому режимі розширення завантажується з локальної папки, в якій міститься файл manifest.json, а також усі скрипти, стилі та HTML-файли, згенеровані Vite. Такий підхід дав змогу швидко інтегрувати веб-застосунок із механізмами браузера, зокрема з popup-вікном розширення.

При розробці веб-застосунку використовувалася система контролю версій Git, що дозволило організувати надійне зберігання коду, відстеження змін та командну роботу в межах проєкту. Репозиторій було створено на платформі GitHub, що забезпечило зручний доступ до історії змін, можливість створення окремих гілок для розробки та ефективний контроль за стабільністю основної версії застосунку. Такий підхід значно спростив тестування нових функцій і дозволив оперативно повертатися до стабільних станів коду у разі потреби. Посилання на GitHub для ознайомленя буде знаходитись у Додатку А.

Таким чином, конструювання розширення передбачає створення повністю автономної системи, що інтегрується безпосередньо в браузер користувача й забезпечує базову функціональність без необхідності підключення до зовнішніх ресурсів.

2.2.1. Реалізація ключових класів

У процесі розробки розширення було виокремлено кілька ключових модулів, які відіграють важливу роль у функціональності системи. Хоча мова програмування JavaScript не завжди передбачає класичну об'єктно-орієнтовану модель (особливо у поєднанні з React), основні функціональні блоки проєкту можна трактувати як логічні класи або функції з чітко визначеними завданнями.

Клас (функція) Transliteratre - цей компонент є центральною частиною розширення, що відповідає за транслітерацію тексту. Його основним завданням є перетворити текст, написаний не тією розкладкою клавіатури, в правильний текст. Наприклад, фраза Ghbdtn буде перетворена на Привіт.

```
export function transliterate(text, direction = 'enToUk') {
  const map = direction === 'enToUk' ? enToUkMap : ukToEnMap;
  let result = '';

  for (let char of text) {
    const lowerChar = char.toLowerCase();
    const mappedChar = map[lowerChar];

    if (mappedChar) {

      result += char === lowerChar ? mappedChar : mappedChar.toUpperCase();
    } else {
      result += char;
    }
  }

  return result;
}
```

Рис. 2.2.1.1 - Клас(функція) Transliteratre

Клас (функція) toggleDirection відповідає за перемикання напрямку транслітерації. Реалізований як частина інтерфейсу (React-кнопка), він оновлює direction у PopupInterface і відображає зміну у UI.

```
const toggleDirection = () => {
  setDirection((prev) => (prev === "enToUk" ? "ukToEn" : "enToUk"));
  setInputText(outputText);
  setOutputText(inputText);
};
```

Рис. 2.2.1.1 - Клас(функція) toggleDirection

Клас (функція) useEffect автоматично оновлює результат виведення. Цей фрагмент коду діє як автоматичний механізм, який стежить за змінами у inputText і direction. Коли одна зі змінних змінюється, функція автоматично викликає transliterate() і оновлює outputText.

```
useEffect(() => {
  const result = transliterate(inputText, direction);
  setOutputText(result);
}, [inputText, direction]);
```

Рис. 2.2.1.1 - Клас(функція) useEffect

Клас (функція) handleCopy виконує копіювання результату. Ця функція дозволяє скопіювати транслітерований текст в буфер обміну за допомогою API navigator.clipboard.

Описані вище фукції є структурою функціонального компонента App що охоплює логіку управління текстом, за допомогою транслітерації, копіюванням і зворотним зв’язком - усе, що потрібно для мінімального, але функціонального браузерного розширення на React.

2.3. Тестування системи та оцінка якості

Після завершення основної розробки розширення було проведено етап тестування з метою перевірки працездатності, надійності та відповідності функціоналу заявленим вимогам. Тестування дозволяє виявити можливі недоліки в роботі програмного продукту, а також оцінити загальну якість реалізації з точки зору кінцевого користувача.

На етапі функціонального тестування перевірялася робота всіх ключових елементів: введення тексту, обробка даних функцією транслітерації, оновлення результату в реальному часі, перемикання напрямку транслітерації та копіювання результату в буфер обміну. Усі дії виконувалися без затримок, і результат відповідав очікуванню. Наприклад, при перемиканні з англійської на українську розкладку текст змінювався миттєво, що підтвердило коректність логіки обробки станів React-компонентів.

Основним типом тестування, застосованим у межах цього проєкту, стало мануальне тестування. Його проведення дало змогу не лише перевірити функціональність окремих частин інтерфейсу, а й оцінити загальний досвід користування. Тестування проводилося за допомогою ручного введення тексту, перевірки реакції інтерфейсу на дії користувача, взаємодії з кнопками та іншими елементами керування. Це дозволило швидко виявити та усунути деякі стилістичні недоліки, які могли погіршити користувацький досвід, зокрема неправильне позиціонування елементів у межах popup-вікна при зміні розміру або масштабування.

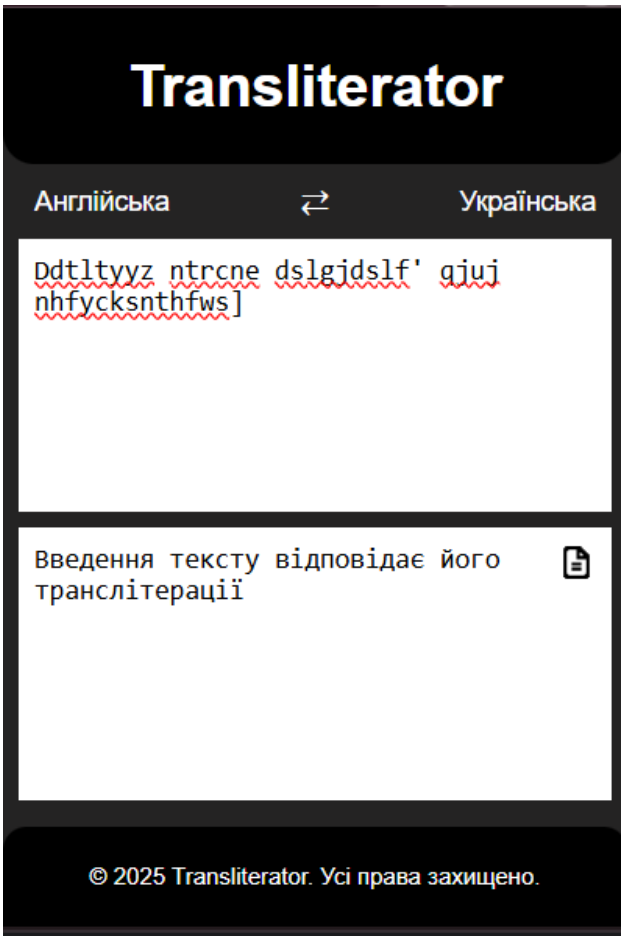

Рис. 2.3.1.1 - Результат тестування транслітерації з англійської на українську.

Окремо проводилося тестування з крайовими випадками — наприклад, введенням порожнього рядка, одного символу, довгого тексту або спеціальних символів. У таких ситуаціях система поводилася передбачувано, не породжувала

помилок і не переривала свою роботу. Тексти з великою кількістю символів оброблялися без втрат продуктивності, що свідчить про ефективність реалізації алгоритму транслітерації.

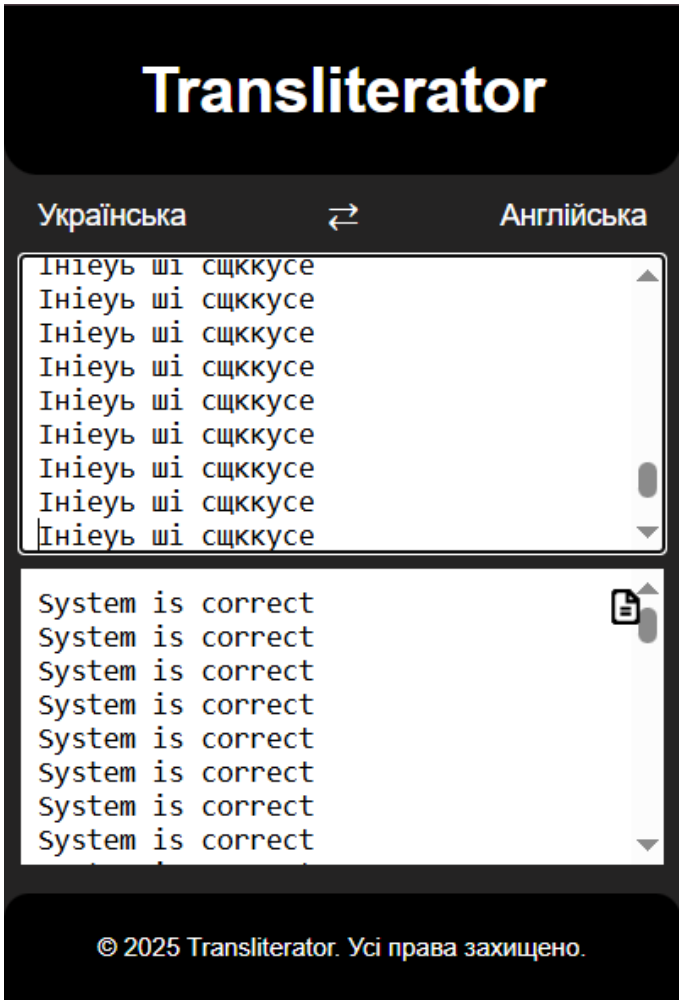

Рис. 2.3.1.1 - Результат тестування транслітерації з української на англійську.

Особливу увагу приділено тестуванню інтерфейсу, зокрема його адаптації до обмеженого простору вікна розширення. Було перевірено правильність відображення всіх елементів: полів введення та результату, кнопки копіювання, перемикача напрямку, а також відповідність стилів при різних масштабах і в темному або світлому режимі браузера. Під час тестування виявлено деякі стилістичні неточності, які було виправлено — зокрема, зміщення кнопок при надто вузькому вікні popup. Також перевірено коректну роботу повідомлення про копіювання тексту: воно з’являється коротко і не заважає користувачу.

Тестування проводилося у декількох браузерах: основним середовищем був Google Chrome, однак окремі перевірки здійснювались у Microsoft Edge, Brave та Opera. Завдяки єдиній основі Chromium усі тести завершилися позитивно, а розширення поводилося стабільно. Різниця спостерігалася лише у зовнішньому вигляді popup в Opera, що обумовлено політикою стилізації самого браузера, але не впливало на функціональність.

Таким чином, після завершення тестування можна стверджувати, що система є стабільною, зручною у використанні та виконує всі передбачені функції без збоїв. Вона витримує як звичайні, так і крайові сценарії використання, має приємний інтерфейс і відповідає вимогам до сучасного браузерного розширення.

3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1. Діяльність. Її види та розуміння в безпеці праці.

Будь-який вид господарської діяльності повинен бути корисний для суспільства, а у сфері бізнесу – ще і рентабельним, але водночас така діяльність може бути джерелом негативних впливів на життя та здоров'я її учасників або інших видів шкоди, наприклад, моральної, що призводять до травматизму, захворювань, повної втрати працездатності чи смерті. Шкоду працівникам може чинити і робота на виробництві (трудова діяльність), і різні види відпочинку, розваги та навіть діяльність, пов'язана з отриманням знань. Історія та практика, таким чином, дають підставу стверджувати, що будь-яка діяльність у сучасних умовах є потенційно небезпечною. Абсолютної безпеки трудової діяльності, тобто стану, в якому виключені всі небезпеки для працівників, просто не існує. Саме тому роботодавцям важливо усвідомити необхідність забезпечення безпеки праці, а без належного методичного інструментарію, розробка якого повинна починатись власне із ідентифікації поняття «безпека праці», реалізація практичних засобів захисту учасників трудового колективу від різного роду небезпек та загроз, не видається можливою.

Люди, знаряддя праці, оточуюче середовище та завдання, що вирішуються в процесі трудової діяльності, являють собою динамічну систему, зміна в якій будь-якого з компонентів веде до зміни інших, а результативний вплив на безпеку інколи буває важко оцінити заздалегідь. Під безпекою розуміється стан захищеності особи та суспільства від ризику зазнати шкоди. Слід зауважити, що між небезпечними (які травмують) і шкідливими виробничими факторами існує певний взаємозв'язок. При високих рівнях ШВФ вони можуть стати небезпечними. Так, досить високі концентрації шкідливих речовин у повітрі робочої зони можуть привести до сильного отруєння і навіть до смерті. Високі рівні звуку або звукового імпульсу можуть привести до травми слухового аналізатору (барабанної перетинки). Високі рівні радіації викликають розвиток

гострої форми променевого захворювання, при якому спостерігається швидке погіршення самопочуття людини з незворотними змінами в організмі, котре приводить, за відсутності медичного втручання, як правило, до смерті. У багатьох випадках наявність у робочий зоні ШВФ сприяє появі НВФ. Наприклад, підвищена вологість й температура, наявність у повітрі робочої зони струмопровідного пилу (шкідливі фактори) значно підвищують небезпеку ураження людини електричним струмом (небезпечний фактор). Таким чином, для ряду негативних факторів поділення на НВФ і ШВФ, деякою мірою, умовне і визначається переважно характером їх прояву у виробничих умовах [16].

Всі фактори на будь-якому підприємстві можуть мати різне походження. Часто можна стикатися з несприятливими умовами праці, які виникають з вини керівництва. Це питання потребує особливої уваги з боку перевіряючих органів. Хочеться сподіватися, що велика частина небезпечних факторів має природне походження, і людині просто необхідно вжити всі заходи, щоб їх вплив був мінімальним. Всі шкідливі виробничі фактори ГОСТ поділяє на наступні групи:

- Фізичні.
- Хімічні.
- Біологічні.
- Психофізіологічні, до яких можна віднести важкі та напружені умови праці.

Діяльність у сфері охорони праці — це системний комплекс заходів, спрямованих на створення безпечних і нешкідливих умов праці, профілактику травматизму та професійних захворювань, а також на забезпечення стабільної роботи підприємства. Така діяльність охоплює кілька основних напрямів: організаційний, технічний, санітарно-гігієнічний, профілактичний та соціально-психологічний. Усі вони взаємодіють між собою для формування єдиного безпечного виробничого середовища.

Стаття 4 Закону України «Про охорону праці» визначає, що засади державної політики в галузі охорони праці базуються на 10 основних принципах, серед яких є пріоритет життя і здоров’я працівників відносно результатів виробничої діяльності, повна відповідальність роботодавця за створення

безпечних умов праці, а також навчання працюючих з питань охорони праці.

3.2 Вплив кольору на покращення умов праці та підвищення продуктивності роботи.

Особливу роль в естетичній організації виробничого середовища відіграє колір, оскільки він справляє багатоплановий вплив на людину, а отже, має багатофункціональне призначення. Колір впливає на фізіологічні і психічні процеси, емоційні стани, працездатність і продуктивність праці працівників. Це зумовлюється такими характеристиками кольору, як колірний тон, насиченість (чистота) і яскравість (відображення світла).

Колірний тон залежить від довжини хвилі, яка вимірюється в мілімікронах. Найдовші хвилі мають червоний та оранжевий кольори. Довжина хвилі фіолетового кольору найменша. Встановлено, що довгохвильові і короткохвильові кольори справляють несприятливий вплив на людину і викликають найбільшу зорову втому. Червоний колір, зокрема, діє як сильний подразник і збуджує нервову систему, а фіолетовий викликає пригнічений настрій. Середньохвильові кольори заспокійливо впливають на нервову систему, сприяють зниженню втоми.

Сила впливу різних кольорів на людину залежить від їх насиченості та яскравості. Насичені кольори покращують настрій і стимулюють роботу аналізаторів, ненасичені та малонасичені кольорові відтінки діють заспокійливо, сприяють зосередженню уваги. Світлі кольори, яскраві і насичені також покращують настрій, а темні — викликають песимістичні настрої. Виходячи з цих властивостей кольори використовують як засіб інформації для орієнтування працівників у виробничому середовищі та устаткуванні з метою дотримання ними техніки безпеки. Стандартом визначені такі значення кольорів: червоний — заборона, безпосередня небезпека; жовтий — попередження, можлива небезпека;

зелений — безпечно; синій — інформація.

Крім цього, кольори мають важливе психофізіологічне значення.

Наведемо приклади впливу деяких кольорів:

- Зелений колір сприяє розслабленню нервової системи і асоціюється з безпекою, стабільністю та відпочинком.
- Синє світло безпосередньо впливає на циркадні ритми людини — воно пригнічує вироблення мелатоніну, що може порушувати режим сну, якщо працівник зазнає впливу такого освітлення у вечірній або нічний час. Водночас синій колір стимулює увагу та підвищує концентрацію в умовах денної активності.
- Червоний, навпаки, викликає емоційне збудження, може сприяти пришвидшенню пульсу та підвищенню артеріального тиску, що важливо враховувати при оформленні приміщень, де потрібна висока точність або знижений рівень стресу [17].

Крім цього, окремі кольори, їх відтінки і поєднання використовуються як додатковий фактор поліпшення освітленості приміщень, для створення необхідного контрасту в полі зору працівника між предметом і фоном, зниження монотонності роботи і втоми, забезпечення психологічного комфорту, підвищення працездатності і продуктивності праці.

Вибір колірного оформлення виробничих приміщень залежить від багатьох факторів — м’язових і нервових навантажень, температурного режиму, розмірів та орієнтації приміщення, монотонності роботи. Так, на роботах, які вимагають великих фізичних і нервових навантажень, а також у цехах з високою темпе-ратурою повітря доцільно використовувати світлі тони голубого, зеленого та інших спокійних холодних кольорів невеликої насиченості. Якщо робота вимагає лише періодичних значних розумових і фізичних навантажень, то вона легше виконується у приміщеннях, пофарбованих у теплі кольори, які підвищують актив-ність організму. Виконання монотонних робіт більш ефективне, якщо приміщення пофарбувати у яскраві кольори, які привертають увагу працівників і розширюють поле коркової активності [18].

ВИСНОВКИ

У межах дипломної роботи було реалізовано браузерне розширення, що дозволяє користувачам швидко транслітерувати текст між англійською та українською розкладками клавіатури. Актуальність теми підтверджується широким використанням текстових інтерфейсів у повсякденній діяльності та частими ситуаціями, коли текст вводиться не тією мовною розкладкою.

Під час розробки було виконано повний цикл створення програмного продукту: від аналізу предметної області та конкурентних рішень до розгортання готового розширення у середовищі браузера Google Chrome. В основі реалізації було використано сучасні вебтехнології, зокрема фреймворк React та збирач Vite, що дозволило забезпечити швидку роботу інтерфейсу та зручність у підтримці коду.

Розширення має адаптований дизайн для обмеженого простору спливаючого вікна, а також реалізує базові функції - перемикання напрямку транслітерації, копіювання результату в буфер обміну та збереження коректного відображення стилів. Продукт протестовано в актуальних версіях браузера, і він показав стабільну роботу за всіх стандартних сценаріїв використання.

Таким чином, поставлені цілі та завдання були успішно реалізовані. Розширення може бути використане як самостійний інструмент, а також слугувати базою для подальшого розвитку - наприклад, додавання підтримки інших мов, автоматичного виявлення розкладки чи інтеграції з онлайн-редакторами.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Зміна розкладки клавіатури. URL: https://design.te.ua/tools/rozkladka/.
2. Виправлення розкладки клавіатури. URL:https://tools.sochka.com/raskladka.html
3. Створюємо та публікуємо розширення. URL:https://dou.ua/forums/topic/48605/
4. Що таке React JS? Як почати вивчати Реакт?
5. Official documentation React. URL: https://react.dev/
6. Детальний огляд та розбір Node.js. URL;https://wezom.com.ua/ua/blog/vse-chto-nuzhno-znat-o-nodejs
7. Introduction to Node.js. URL: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs
8. Чому JavaScript — перспективна мова програмування? URL:https://dou.ua/forums/topic/35184/
9. Ключові методології розробки програмного забезпечення. URL:https://wezom.com.ua/ua/blog/metodologija-razrabotki-programmnogo-obespechenija
10. What is the Waterfall methodology? URL:https://business.adobe.com/blog/basics/waterfall#:~:text=The%20Waterfall%20methodology%20%E2%80%94%20also%20known%20as,wrapping%20up%20before%20the%20next%20phase%20begins.&text=The%20Waterfall%20methodology%20depends%20on%20the%20belief,requirements%20can%20be%20gathered%20and%20understood%20upfront.
11. What is Scrum? URL: https://www.scrum.org/resources/what-scrum-module
12. Основні типи архітектури програмного забезпечення. URL:https://www.artofba.com/uk/post/main-types-of-software-architecture
13. Single Page Application (SPA). URL:https://romul.name/single-page-application-spa/

14. Що таке Single Page Application? URL:https://asabix.com.ua/what-is-a-single-page-application/

15. Manifest V3. URL: https://developer.chrome.com/docs/extensions/develop/migrate/what-is-mv3

16. Про охорону праці : Закон України від 14.10.1992 № 49 URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text

17. НАЦІОНАЛЬНИЙ СТАНДАРТ УКРАЇНИ: Графічні символи КОЛЬОРИ ТА ЗНАКИ БЕЗПЕКИ. Частина 1. Принципи проектування знаків безпеки від 25.05.2005 № 128 з 2006- 10-01 URL: :https://zakon.isu.net.ua/sites/default/files/pdf/grafichni_simvoli._kolori_ta_zna-3-30608.pdf

18. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин від від 10.12.98 № 7 URL:https://zakon.rada.gov.ua/rada/show/v0007282-98#Text

19. Дистанційний курс «Кваліфікаційні роботи бакалаврів» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: URL: https://dl.tntu.edu.ua/bounce.php?course=5329

20. Методичні вказівки до виконання дипломної роботи освітнього рівня - бакалавр студентами усіх форм навчання для напряму підготовки 121 – Інженерія програмного забезпечення/ Укладачі : Петрик М.Р.,Михалик Д.М., Кінах Я.І., Гладьо С.В., Цуприк Г.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 28 с

21. Інформаційні моделі, системи та технології Тернопільського національного технічного університету імені Івана Пулюя, Яковенко, І. Мудрик. ТНТУ, 2022.С.181- 182.

ДОДАТКИ

ДОДАТОК А.

Лістинг файлу App.jsx (основний інтерфейс користувача):

```
import { useState, useEffect } from "react";
import { transliterate } from "./translit";
import "./index.css";
import copyIcon from "./assets/copy-icon.svg";

function App() {
  const [inputText, setInputText] = useState("");
  const [outputText, setOutputText] = useState("");
  const [direction, setDirection] = useState("enToUk");
  const [copied, setCopied] = useState(false);

  useEffect(() => {
    const result = transliterate(inputText, direction);
    setOutputText(result);
  }, [inputText, direction]);

  const toggleDirection = () => {
    setDirection((prev) => (prev === "enToUk" ? "ukToEn" :
"enToUk"));
    setInputText(outputText);
    setOutputText(inputText);
  };

  const handleCopy = () => {
    navigator.clipboard.writeText(outputText);
    setCopied(true);
    setTimeout(() => setCopied(false), 1500);
  };

  return (
    <div className="app-container">
      <header className="app-header">
        <h1 className="header-name">Transliterator</h1>
      </header>

      <main className="app-main">
        <div className="language-labels">
          <span>{direction === "enToUk" ? "Англійська" :
"Українська"}</span>
          <button className="toggle-button"
onClick={toggleDirection}>
    ⇄
          </button>
```

```
          <span>{direction === "enToUk" ? "Українська" :
"Англійська"}</span>
        </div>
        <div className="textareas">
          <textarea
            placeholder="Введіть текст..."
            value={inputText}
            onChange={(e) => setInputText(e.target.value)}
          />
          <div className="output-wrapper">
            <textarea
              placeholder="Виправлений текст..."
              value={outputText}
              readOnly
            />
            <button
              className="copy-button"
              onClick={handleCopy}
              title="Копіювати"
            >
              <img
                src={copyIcon}
                alt="Copy"
                style={{ width: "20px", height: "20px" }}
              />
            </button>

            {copied && <span className="copied-
label">Скопійовано!</span>}
          </div>
        </div>
      </main>

      <footer className="app-footer">
        <p>© 2025 Transliterator. Усі права захищено.</p>
      </footer>
    </div>
  );
}

export default App;
```

Лістинг файлу keyboardMap.js (відповідність символів між розкладками клавіатур):

```
export const engToUkrMap = {
  'q': 'й', 'w': 'ц', 'e': 'у', 'r': 'к', 't': 'е',
  'y': 'н', 'u': 'г', 'i': 'ш', 'o': 'щ', 'p': 'з',
  '[': 'х', ']': 'ї',
  'a': 'ф', 's': 'і', 'd': 'в', 'f': 'а', 'g': 'п',
  'h': 'р', 'j': 'о', 'k': 'л', 'l': 'д', ';': 'ж', "'": 'є',

  'z': 'я', 'x': 'ч', 'c': 'с', 'v': 'м', 'b': 'и',
  'n': 'т', 'm': 'ь', ',': 'б', '.': 'ю', '/': '.',

  '`': '’', ' ': ' '
};

export const enToUkMap = {
  'q': 'й', 'w': 'ц', 'e': 'у', 'r': 'к', 't': 'е',
  'y': 'н', 'u': 'г', 'i': 'ш', 'o': 'щ', 'p': 'з',
  '[': 'х', ']': 'ї', 'a': 'ф', 's': 'і', 'd': 'в',
  'f': 'а', 'g': 'п', 'h': 'р', 'j': 'о', 'k': 'л',
  'l': 'д', ';': 'ж', "'": 'є', 'z': 'я', 'x': 'ч',
  'c': 'с', 'v': 'м', 'b': 'и', 'n': 'т', 'm': 'ь',
  ',': 'б', '.': 'ю', '/': '.', '`': '’'
};

export const ukToEnMap = Object.fromEntries(
  Object.entries(enToUkMap).map(([en, uk]) => [uk, en])
);
```

Лістинг файлу translit.js (логіка транслітерації введеного тексту):

```
import { enToUkMap, ukToEnMap } from './keyboardMap';
export function transliterate(text, direction = 'enToUk') {
  const map = direction === 'enToUk' ? enToUkMap : ukToEnMap;
  let result = '';
  for (let char of text) {
    const lowerChar = char.toLowerCase();
    const mappedChar = map[lowerChar];
    if (mappedChar) {
      result += char === lowerChar ? mappedChar :
mappedChar.toUpperCase();
    } else {
      result += char;
    }
```

```
    }
    return result;
}
```

1. Репозиторій на GitHub. URL: https://github.com/Ittero/Transliterator_extension.git

ДОДАТОК Б. Тези конференції.

УДК 004.4

Д. Буряк, І. Мудрик
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ПІДХОДИ ДО РЕАЛІЗАЦІЇ ФУНКЦІОНАЛУ ПЕРЕВІРКИ ТЕКСТУ У БРАУЗЕРНОМУ СЕРЕДОВИЩІ

UDC 004.4
D. Buriak, I. Mudryk

APPROACHES TO IMPLEMENTING TEXT VERIFICATION FUNCTIONALITY IN THE BROWSER ENVIRONMENT

У наш час текстова інформація є одним із ключових засобів комунікації в цифровому просторі. Щоденно мільйони користувачів взаємодіють із веб-застосунками, створюючи текстовий контент: публікації, повідомлення, електронні листи, коментарі тощо. Проте, незважаючи на широкий спектр платформ, далеко не всі вони мають вбудовані інструменти для перевірки орфографії, граматики чи стилістики. Як результат — зниження якості контенту, втрати довіри до ресурсу або навіть викривлення змісту через незначні помилки. Ця проблема є особливо актуальною в умовах зростання кількості онлайн-комунікацій та необхідності дотримання мовних стандартів. Відсутність перевірки тексту може також негативно впливати на професійний імідж як окремих користувачів, так і компаній загалом.

Одним з ефективних рішень цієї проблеми є розробка розширень для браузера, які б дозволяли автоматично перевіряти та виправляти текст безпосередньо в інтерфейсі користувача. Такий підхід забезпечує універсальність, оскільки дозволяє інтегруватися з будь-яким вебсайтом, не змінюючи його вихідний код. Основна мета проєкту полягає у створенні функціонального, гнучкого та ефективного розширення, здатного аналізувати введений текст у режимі реального часу, виявляти помилки та пропонувати користувачу можливі варіанти виправлення. Такий інструмент може суттєво підвищити зручність написання тексту, особливо у середовищах, де відсутні вбудовані системи перевірки.[3][4]

Важливу роль у реалізації проєкту відіграє технологія Node.js — середовище виконання JavaScript-коду на стороні серверу. У межах проєкту Node.js використовується для створення серверної частини, яка відповідає за обробку тексту, взаємодію з зовнішніми API (наприклад, сервісами перевірки граматики) та формування відповіді для клієнта. Завдяки асинхронній моделі Node.js забезпечується швидка обробка запитів, що є критичним для забезпечення роботи розширення в реальному часі. Крім того, використання спільної мови програмування для клієнтської та серверної частин (JavaScript) спрощує розробку і підтримку проєкту.[2]

Для побудови інтерфейсу розширення використовується бібліотека React. Вона дозволяє створити динамічний UI, який адаптується під дії користувача. React забезпечує зручну роботу з компонентами, станом додатка та ефективне оновлення інтерфейсу, що робить його ідеальним для інтеграції з браузерним середовищем, зокрема, для створення інтерактивного вікна з підказками, налаштуваннями, підсвіткою помилок тощо. Такий підхід дозволяє реалізувати зручну та інтуїтивно зрозумілу взаємодію з користувачем навіть у складних сценаріях перевірки тексту.[1]

Під час реалізації розширення також враховано питання безпеки, продуктивності та зручності використання. Завдяки поєднанню Node.js і React є можливість досягти балансу між продуктивністю на сервері та зручністю користувацького інтерфейсу.

Для ефективної перевірки тексту сучасні розширення браузера активно використовують зовнішні API, зокрема ті, що базуються на технологіях штучного інтелекту. Саме така інтеграція API надає можливість проводити глибокий аналіз тексту з урахуванням граматики, пунктуації, стилістики й навіть контексту вживання слів. Завдяки цьому перевірка стає не лише формальною, а й змістовною, що особливо важливо при створенні складних або професійних текстів. Інтеграція

подібних сервісів у розширення здійснюється через HTTP-запити, а результати аналізу повертаються у структурованому форматі (наприклад, JSON), що спрощує їх обробку на клієнтській стороні.

Список використаних джерел:

1. Cases. Що таке React JS? Як почати вивчати Реакт? Навички для React Developer: https://cases.media/article/sho-take-react-js-yak-pochati-vivchati-reakt-navichki-dlya-react-developer?srsltid=AfmBOoq-mjrXdyqqgfB5sVg65Pl0ZEwNRb73oOKQlp3mSaQqjMYSb5YP
2. Node.js. About Node.js: https://nodejs.org/en/about
3. Towardsdatascience. Grammar Error Correction: https://towardsdatascience.com/grammar-error-correction-af365dad794/
4. Machine learning models and methods aspects of processing unstructured data. Bryk, I Mudryk, M Holubovskyi, Y Stoianov. (BAIT 2024) Zboriv, Ukraine, 2024. pp. 64-74: https://ceur-ws.org/Vol-3842/paper4.pdf
5. Awe Amason. What is an API (Application Programming Interface)?: https://aws.amazon.com/what-is/api/

ДОДАТОК Г. Диск з роботою.