МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МАРІУПОЛЬСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЕКОНОМІКО-ПРАВОВИЙ ФАКУЛЬТЕТ
КАФЕДРА СИСТЕМНОГО АНАЛІЗУ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

До захисту допустити:
В.о. зав. кафедри

Ганна МАРТИНЮК

«22» листопада 2024 р.

«АНАЛІЗ ВПЛИВУ UI БІБЛІОТЕК НА РОЗРОБКУ ДОДАТКІВ»

Кваліфікаційна робота
здобувача вищої освіти другого (магістерського) рівня вищої освіти освітньо-професійної програми «Системний аналіз»
Шовчко Феоктиста Дмитровича
Науковий керівник:
Мартинюк Ганна Вадимівна,
кандидат технічних наук, доцент,
в.о. завідувача кафедри системного аналізу та інформаційних технологій
Рецензент:
Монченко Олена Володимирівна,
кандидат технічних наук, доцент, професор кафедри біокібернетики та аерокосмічної медицини Національного авіаційного університету

Кваліфікаційна робота захищена
з оцінкою відмінно 96 (А)
Секретар ЕК

«16» грудня 2024 р.

Київ – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

UI	User Interface (Користувацький інтерфейс)
UX	User Experience (Користувацький досвід)
HTML	Hyper Text Markup Language (Мова розмітки гіпертексту)
CSS	Cascading Style Sheets (Каскадні таблиці стилів)
DOM	Document Object Model (Об'єктна модель документа)
API	Application Programming Interface (інтерфейс прикладного програмування)
E2E	End-to-End (повне тестування від кінця до кінця)
ARIA	Accessible Rich Internet Applications (Доступні насичені інтернет-застосунки)

ВСТУП

Розвиток інформаційних технологій, що охоплює широке коло сфер - від наукових досліджень та медицини до розважальної індустрії та бізнесу, визначає зростаючі вимоги до якості, швидкодії та ефективності розробки програмних продуктів. Вебдодатки та мобільні застосунки поступово стають невід'ємною частиною інформаційного суспільства, оскільки вони забезпечують оперативний доступ до різноманітних сервісів, платформ та даних, сприяючи підвищенню комфорту та ефективності взаємодії користувачів з цифровим середовищем. У цьому контексті набуває особливої ваги оптимізація процесів розробки інтерфейсів користувача, які слугують ключовим засобом взаємодії між користувачем і системою.

Одним із визначальних чинників, що впливає на якість і швидкість розробки користувацьких інтерфейсів, є застосування готових рішень для стилізації додатків, зокрема, бібліотек інтерфейсу користувача. Такі бібліотеки надають набір готових компонентів, включно з кнопками, формами, навігаційними меню, каруселями, анімаціями та іншими елементами, які можуть бути швидко інтегровані в нові або існуючі проєкти. Це дозволяє розробникам зменшити витрати часу на створення базових компонентів інтерфейсу та зосередитися на розробці унікальних функціональних можливостей додатка та його загальному дизайні. Застосування UI-бібліотек сприяє стандартизації процесу розробки, забезпечуючи уніфіковані підходи до створення інтерфейсів, що відповідають сучасним тенденціям дизайну та очікуванням користувачів.

Актуальність дослідження визначається необхідністю оцінки впливу різних UI бібліотек на процес розробки додатків, включаючи фактори продуктивності, підтримки, розширюваності та користувацького досвіду.

Об'єктом дослідження є процеси розробки програмних додатків із використанням UI бібліотек.

Предметом дослідження є методи та засоби впровадження та використання UI бібліотек у розробці додатків.

Метою даної роботи є аналіз впливу UI бібліотек на розробку додатків та розробка власної бібліотеки інтерфейсу користувача для вебсайтів, що забезпечить більш гнучкий та ефективний процес розробки. Для досягнення поставленої мети у дослідженні було поставлено наступні завдання:

- провести теоретичний аналіз існуючих UI бібліотек, їх переваг та недоліків;

- визначити вплив UI бібліотек на процес розробки додатків та користувацький досвід;

- розробити власну UI бібліотеку для вебсайтів, використовуючи сучасні підходи та технології;

- оцінити ефективність запропонованого рішення шляхом тестування та порівняння з існуючими UI бібліотеками.

Наукові публікації та дослідження пропонують різні підходи до оцінки ефективності UI-бібліотек, однак існує потреба у глибшому аналізі та порівнянні їхнього впливу на розробку додатків, враховуючи різні типи проєктів та умови використання. Зокрема, в роботі Джонатана Андерсона [15] детально досліджується створення ефективного користувацького досвіду, що є важливою складовою UI-бібліотек. Це джерело допомагає оцінити вплив UI на користувацький досвід (UX) і надає інструменти для створення продуктивних інтерфейсів. Також Ніколас Клауд [6] аналізує сучасні JavaScript-фреймворки, які часто інтегруються з UI-бібліотеками, що дозволяє краще зрозуміти їхній вплив на продуктивність і можливості адаптації. Аравінд Шеной [2] зосереджується на використанні Bootstrap -

однієї з найпопулярніших UI-бібліотек, розглядаючи її переваги та обмеження, що є ключовими факторами при виборі бібліотеки для різних проєктів.

Результати кваліфікаційної роботи були представлені в моїх тезах "Impact Analysis of UI Libraries on Developer and User Experience" на пленарному засіданні ЕПФ 2024 [23], де було розглянуто практичні аспекти впливу UI-бібліотек на процес розробки і досвід користувачів.

Наукова новизна роботи полягає у розробці власної UI бібліотеки, яка поєднує гнучкість, адаптивність та простоту використання, а також у визначенні критеріїв ефективності використання UI бібліотек у різних контекстах.

Практичне значення отриманих результатів полягає у можливості їх застосування в реальних проєктах для підвищення продуктивності розробки шляхом скорочення часу на створення та тестування інтерфейсів, покращення користувацького досвіду через використання сучасних і зручних компонентів, а також зменшення витрат на підтримку та розширення програмних продуктів завдяки модульності та масштабованості бібліотеки. Крім того, розроблена бібліотека може бути використана як основа для створення нових продуктів або адаптації до специфічних вимог клієнтів. Завдяки роботі команди проєкт був успішно інтегрований на платформу “Hewlett Packard Enterprise”, а також привернув увагу представників компанії Microsoft, що свідчить про його потенціал і перспективи для подальшого використання в галузі.

Очікується, що результати дослідження можуть бути впроваджені у компаніях, які займаються розробкою програмного забезпечення, з метою оптимізації процесів створення інтерфейсів користувача. Вони також можуть бути корисними у навчальному процесі для підготовки фахівців з

інформаційних технологій, надаючи їм сучасні інструменти та методології для ефективної роботи в галузі розробки програмного забезпечення.

РОЗДІЛ 1
ТЕОРЕТИЧНІ АСПЕКТИ ВИКОРИСТАННЯ UI БІБЛІОТЕК У РОЗРОБЦІ ДОДАТКІВ

1.1 Визначення та класифікація UI бібліотек

У сучасних умовах розробка вебдодатків та мобільних застосунків стає дедалі складнішою та більш вимогливою. Зростаюча потреба у швидкому створенні функціональних, зручних і привабливих інтерфейсів користувача висуває нові вимоги до підходів та інструментів, що застосовуються розробниками. Правильно спроєктований користувацький інтерфейс є не лише естетичним оформленням елементів, таких як кнопки чи панелі, але й важливим чинником загального користувацького досвіду, що визначає, наскільки інтуїтивно, ефективно та комфортно користувач взаємодіє з додатком або вебсайтом.

Створення вебсайтів передбачає використання різноманітних компонентів, що складають UI і забезпечують взаємодію користувача із сайтом. Ці компоненти, такі як кнопки, поля вводу, навігаційні панелі та форми, пишуться за допомогою мови розмітки HTML. HTML є основою для структурування вебсторінок і надає базові елементи для створення простих інтерфейсів, однак стандартні HTML компоненти мають обмежену функціональність і зовнішній вигляд, що часто не задовольняє сучасні вимоги до дизайну та UX.

Щоб підвищити якість та адаптивність інтерфейсів, розробники все частіше звертаються до бібліотек інтерфейсу користувача (UI бібліотек). Ці бібліотеки містять набори готових компонентів, таких як кнопки, форми, навігаційні панелі, таблиці, модальні вікна та інші елементи, які можна

використовувати для побудови інтерфейсів користувача. Як зазначається у праці [1, с.44], UI бібліотеки дозволяють розробникам зосередитися на створенні інноваційних функцій і поліпшенні користувацького досвіду, зменшуючи потребу в ручному створенні компонентів з нуля.

Бібліотеки значно спрощують розробку та підтримку програмних продуктів завдяки своїй стандартизації та повторюваності. Вони забезпечують єдину мову дизайну, що дозволяє зменшити кількість помилок, покращити взаємодію з користувачем та підвищити загальну ефективність розробки. Окрім цього, вони сприяють зниженню витрат часу та ресурсів, оскільки більшість елементів уже протестовані та оптимізовані для використання на різних платформах і пристроях.

Розвиток UI бібліотек як окремого напрямку програмного забезпечення тісно пов'язаний з еволюцією вебтехнологій і розширенням можливостей браузерів. У перші роки Інтернету, коли HTML тільки почав набувати популярності, розробники використовували основні HTML-теги для створення статичних сторінок, а стилізація обмежувалась базовими CSS правилами. Перші спроби стандартизувати користувацькі інтерфейси почалися з появою бібліотек JavaScript, таких як jQuery, яка спрощувала взаємодію з DOM та підтримувала кросбраузерність.

З розвитком Інтернету та технологій зросла потреба у створенні більш складних і динамічних інтерфейсів користувача. Відповідно до цієї потреби, з'явилися нові бібліотеки, кожна з яких пропонувала унікальні рішення і особливості. Одним із значних досягнень у цій сфері стала поява бібліотеки “Bootstrap” у 2011 році, яка стала однією з перших UI бібліотек, що дозволила розробникам швидко створювати адаптивні та сучасні вебдизайни. Bootstrap популяризувала концепцію "адаптивного дизайну" [2], що забезпечує автоматичну адаптацію інтерфейсу до різних розмірів екранів, яке

значно спростило процес розробки вебсайтів, оптимізованих для різних пристроїв.

З моменту появи перших бібліотек для інтерфейсів користувача цей напрямок розробки зазнав значного розширення. З'явилися нові рішення, які надають розробникам можливість створювати застосунки швидше і з меншими ресурсними затратами. Це сприяло інтенсивному розвитку індустрії, де різноманіття рішень відкриває нові горизонти для розробки, адаптуючи інтерфейси до різноманітних вимог і стандартів. Збільшення кількості доступних UI-бібліотек внесло вагомий внесок у стандартизацію та оптимізацію процесів розробки, що зробило їх більш передбачуваними, гнучкими та ефективними.

Загалом UI бібліотеки можна класифікувати за типом застосування, що визначає їхню спеціалізацію та область ефективності, відповідно до специфічних потреб і характеристик середовищ, у яких вони використовуються. Цей підхід дозволяє систематизувати різні типи бібліотек на основі їхньої функціональності і призначення [3, с.106].

Зокрема, основною класифікацією є поділ за призначенням, що визначає область застосування бібліотеки та її відповідність конкретним потребам розробки:

- Веб-бібліотеки, призначені для роботи в браузерах, надають компоненти, такі як навігаційні панелі, кнопки, таблиці та форми, що значно спрощують розробку та створення функціональних вебінтерфейсів. Прикладами таких бібліотек є Bootstrap, Material-UI, Semantic UI та Foundation.

- Мобільні UI бібліотеки орієнтовані на розробку інтерфейсів для мобільних застосунків і часто розробляються для конкретних платформ, таких як iOS або Android. Наприклад, SwiftUI є популярною бібліотекою для iOS, тоді як Jetpack Compose є аналогічним рішенням для Android.

- Гібридні UI бібліотеки забезпечують можливість створення інтерфейсів як для вебдодатків, так і для мобільних платформ, використовуючи один код. Це дозволяє розробникам зменшити витрати часу і зусиль на розробку окремих версій для різних платформ. Прикладами таких бібліотек є React Native і Flutter, які популярні серед розробників завдяки своїй здатності підтримувати кросплатформені рішення.

Класифікація за функціональністю визначає набір можливостей UI-бібліотек та ступінь складності компонентів, які вони надають:

- Базові бібліотеки компонентів містять мінімальний набір основних компонентів, таких як кнопки, поля введення, форми, навігація. Вони легкі та швидкі у використанні, але обмежені в можливостях. Прикладами таких бібліотек є Bulma або Pure CSS.

- Розширені бібліотеки компонентів включають велику кількість компонентів, модулів і інструментів для побудови більш складних інтерфейсів. До них належать Material-UI, Ant Design, Chakra UI, які надають додаткові функції, як-от темізація, керування станами, інтеграція з іншими бібліотеками.

Класифікація UI-бібліотек також відбувається за технологічною платформою, яка визначає їх інтеграцію з певним стеком технологій і фреймворками:

- JavaScript UI бібліотеки орієнтовані на використання в екосистемі JavaScript і зазвичай інтегруються з популярними фреймворками, такими як React, Vue.js або Angular. Наприклад, Material-UI є бібліотекою, спеціально розробленою для React, і надає компоненти, що відповідають принципам Material Design, забезпечуючи однорідний вигляд і функціональність інтерфейсу.

- CSS UI бібліотеки зосереджені на стилізації інтерфейсу і надають готові CSS-класи для компонування елементів без залежності від конкретних JavaScript-фреймворків. Вони можуть бути використані в поєднанні з будь-якими технологіями JavaScript, що забезпечує їхню універсальність. Прикладами таких бібліотек є Bootstrap і Foundation, які забезпечують зручний спосіб створення стильних і функціональних інтерфейсів.

- Гібридні бібліотеки поєднують у собі як JavaScript, так і CSS-компоненти, що забезпечує гнучкість у використанні в різних контекстах і технологічних середовищах. Наприклад, Semantic UI інтегрує компоненти для стилізації разом з JavaScript-скриптами, які забезпечують додаткову інтерактивність. Це дозволяє розробникам отримати комплексне рішення для створення інтерактивних і візуально привабливих інтерфейсів, незалежно від специфіки технологічного стека.

За принципом ліцензування UI-бібліотеки класифікуються залежно від умов їх використання та можливості модифікації:

- Відкриті (Open Source) бібліотеки доступні для безкоштовного використання та модифікації, що робить їх популярними серед розробницьких спільнот завдяки високій гнучкості та можливості внесення змін відповідно до потреб користувачів. Такі бібліотеки зазвичай підтримуються активними спільнотами розробників і мають широке застосування. Прикладами є React, який є бібліотекою для побудови інтерфейсів користувача, Material-UI, яка надає компоненти відповідно до принципів Material Design, та Bootstrap, що забезпечує готові рішення для стилізації веб-додатків.

- Закриті (Proprietary) бібліотеки, в свою чергу, використовуються в комерційних цілях і зазвичай вимагають придбання ліцензії для легального використання. Ці бібліотеки можуть пропонувати додаткову підтримку, розширені можливості та спеціалізовані функціональності, що відповідають

потребам підприємств і організацій. Прикладами таких бібліотек є Kendo UI і Sencha Ext JS, які надають професійні рішення з високим рівнем підтримки і додатковими інструментами для розробки складних інтерфейсів.

Відомий блогер Тео Браун у своєму нещодавньому відео запропонував власну класифікацію UI бібліотек, розділивши їх на декілька типів [4]:

- Розширення CSS, такі як Sass, Less, Tailwind, CSS Modules, дозволяють створювати стиль інтерфейсу на власний розсуд.

- Бібліотеки поведінки (Headless), наприклад, HeadlessUI, Radix, React Aria, визначають поведінку компонентів, але не їхній візуальний стиль.

- Системи стилізації, такі як TailwindUI, DaisyUI, надають готовий вигляд і поведінку, але з можливістю налаштування.

- Бібліотеки компонентів, наприклад, MUI, Ant Design, Mantine, які мають власний набір правил і компонентів, і вимагають певного навчання для їх ефективного використання.

Класифікація UI бібліотек є важливим етапом у виборі найвідповіднішого інструменту для конкретного проєкту. Вона допомагає розробникам визначити оптимальне рішення залежно від типу застосування, функціональних потреб і технологічної платформи. Вибір відповідної бібліотеки має суттєвий вплив на архітектуру додатка, процес розробки, тестування та подальшу підтримку продукту [5, с.165]. Проте, поряд із класифікацією, також важливо враховувати якість бібліотеки, яка може бути оцінена за кількома критеріями (табл. 1.1).

Таблиця 1.1

Критерії оцінки якості UI-бібліотеки

Критерій оцінки	Опис
Послідовність елементів UI, шаблонів і поведінки	Забезпечення єдиної стилістики та функціональності в межах застосунку.
Персоналізація та розширення	Можливість адаптації бібліотеки для специфічних потреб різних застосунків.
Оптимізація для рендерингу та малої ваги	Висока продуктивність та мала вага компонентів, що є критичним для мобільних пристроїв.
Відповідність стандартам вебдоступності	Дотримання стандартів WCAG для забезпечення доступу до веб-контенту для людей з різними типами інвалідності.
Документація та приклади	Наявність якісної документації та прикладів для спрощення навчання та швидкого впровадження.
Працездатність у сучасних браузерах	Сумісність з сучасними браузерами та підтримка поступового покращення для старих версій браузерів.
Активна спільнота	Наявність активної спільноти, яка сприяє розвитку бібліотеки та надає підтримку користувачам.

Продовження таблиці 1.1

Відповідність очікуванням користувачів та розробників	Бібліотека має бути інтуїтивно зрозумілою і відповідати очікуванням кінцевих користувачів та розробників.
Підтримка адаптивних дизайнів	Забезпечення адаптивності інтерфейсів для різних розмірів екранів і пристроїв.

Вибір UI бібліотеки є критично важливим етапом у розробці веб- та мобільних застосунків. Розробники повинні відповідально підійти до цього процесу, враховуючи різні фактори, такі як тип застосування, функціональність, технологічна платформа і принципи ліцензування. Важливо не лише вибрати відповідну бібліотеку, але й ретельно оцінити її якість, зокрема з огляду на послідовність, гнучкість, продуктивність, доступність та інші критерії. Комплексний підхід до вибору і оцінки бібліотеки допоможе забезпечити ефективність, зручність і довговічність розробленого інтерфейсу.

1.2. Огляд популярних UI бібліотек

У сучасній веброзробці використання бібліотек інтерфейсу користувача (UI бібліотек) набуло ключового значення для створення інтерактивних, естетично привабливих і функціональних вебсайтів та застосунків. Вони надають широкий набір компонентів, що скорочує час розробки, підвищує продуктивність і спрощує впровадження стандартів дизайну. Важливим аспектом є можливість стандартизації інтерфейсу та забезпечення його адаптивності до різних пристроїв, що особливо актуально в умовах розмаїття

сучасних екранів і платформ. До того ж, UI бібліотеки спрощують дотримання вимог доступності, що сприяє створенню інклюзивних продуктів.

Розвиток технологій веброзробки охоплює як прості засоби, такі як мова програмування JavaScript або її надбудова TypeScript, так і більш складні інструменти, такі як фреймворки.

Для спрощення роботи з великими проєктами використовуються фреймворки, такі як React або Vue.js. Вони забезпечують розширені можливості для структуризації коду та впровадження сучасних архітектурних підходів. Фреймворк - це не просто набір інструментів, а програмна платформа, яка формує основу для розробки застосунків, встановлюючи чіткі стандарти й правила. Це дозволяє зменшити кількість коду та уникнути дублювання, що прискорює процес створення програмного продукту.

Фреймворки зазвичай містять набір готових компонентів і функцій, що можуть використовуватися повторно для вирішення типових завдань. Наприклад, фреймворк для веброзробки може містити бібліотеки для обробки запитів HTTP, маніпуляції даними, інтеграції з базами даних та відображення інформації на сторінці. Це дозволяє розробникам зосередитись на бізнес-логіці та унікальних особливостях своїх застосунків, а не на вирішенні загальних проблем.

На відміну від фреймворків, UI бібліотеки спеціалізуються на створенні компонентів інтерфейсу користувача, таких як кнопки, форми, навігаційні меню, модальні вікна тощо. Вони зосереджені на візуальній та функціональній складовій інтерфейсу користувача, надаючи готові рішення для створення сучасних, адаптивних і привабливих інтерфейсів. Наприклад, бібліотеки можуть включати компоненти для створення таблиць даних, форм

введення, індикаторів прогресу тощо, які відповідають найкращим практикам дизайну та доступності.

UI бібліотеки часто використовуються у поєднанні з фреймворками, такими як раніше згадані React або Vue, щоб покращити розробницький процес і забезпечити узгодженість та високу якість інтерфейсів. Використання фреймворку дає можливість швидко налаштувати основні функціональні частини вебзастосунку, тоді як UI бібліотеки додають готові інструменти для побудови користувацького інтерфейсу, знижуючи потребу в написанні власних компонентів з нуля.

Фреймворки, часто сприймаються як бібліотеки для створення інтерфейсів користувача, однак їхня основна мета полягає в забезпеченні інструментів для побудови компонентів, що визначають як поведінку, так і структуру застосунків. Як зазначає Ніколас Клауд [6], "фреймворки забезпечують не лише набір компонентів, а й створюють архітектурну основу, що дозволяє розробляти масштабовані та стійкі додатки". Важливою особливістю таких платформ є підтримка компонентно-орієнтованої архітектури, яка дозволяє створювати вебзастосунки, що складаються з незалежних, легко повторно використовуваних частин.

React, зокрема, застосовує концепцію віртуальної DOM (Document Object Model), що значно підвищує ефективність оновлення та рендерингу вебсторінок, забезпечуючи швидший і динамічніший користувацький інтерфейс. DOM - це структура даних, створювана браузером на основі HTML-документа, яка перетворює його в ієрархічне дерево елементів, таких як заголовки, параграфи, зображення та кнопки. Це дерево відображає всю структуру сторінки й слугує основою для взаємодії користувача з елементами інтерфейсу (рис. 1.1).

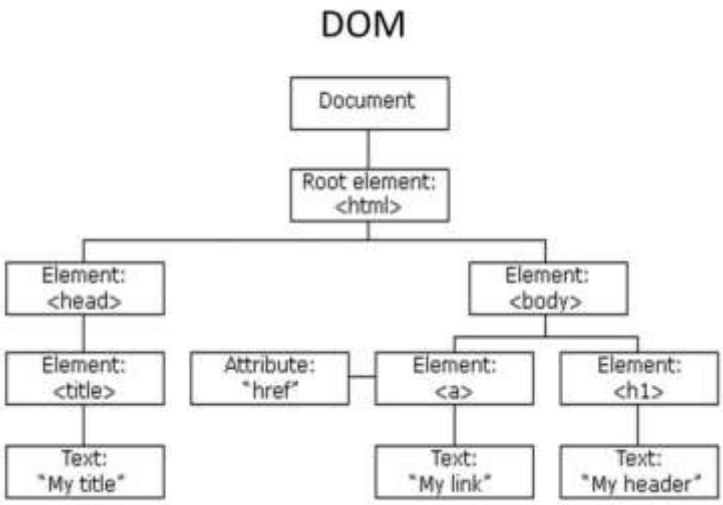


Рисунок 1.1: Document Object Model

Оновлення реальної DOM може бути повільним, оскільки навіть невеликі зміни на сторінці можуть викликати необхідність перемалювати велику частину дерева, що створює додаткове навантаження на систему. Віртуальна DOM у React вирішує цю проблему шляхом створення полегшеної копії реальної DOM, з якою React працює у пам'яті. При кожній зміні на сторінці віртуальна DOM оновлюється спершу, порівнюючи нову та стару версії дерева, і лише після цього реальна DOM змінюється мінімально необхідним чином. Це значно знижує кількість операцій оновлення та підвищує загальну продуктивність інтерфейсу, що є особливо важливим у випадку великих та складних вебзастосунків, де продуктивність є критичним фактором [7].

Зважаючи на різні підходи до розробки інтерфейсів, можна виділити два основних типи UI бібліотек: універсальні бібліотеки, які можуть використовуватися з будь-якою мовою програмування або фреймворком, та бібліотеки, спеціально створені для роботи з певними фреймворками. Однією з найстаріших універсальних бібліотек є jQuery-UI. Вона побудована на основі популярної JavaScript бібліотеки jQuery, яка спрощує взаємодію з HTML-документами, обробку подій, анімацію та AJAX-запити.

jQuery-UI надає розширений набір функціональності, включаючи елементи керування інтерфейсом користувача, такі як діалогові вікна, кнопки, автозаповнення, слайдери та багато інших [8]. Хоча бібліотека jQuery-UI не так активно використовується в нових проєктах через втрати своєї актуальності, вона залишається корисним інструментом для підтримки старих проєктів і простих вебзастосунків (рис. 1.2).

Рисунок 1.2: Логотип JQuery

Однією з найпопулярніших і найбільш використовуваних бібліотек інтерфейсу користувача є Bootstrap, яка була розроблена інженерами компанії Twitter і офіційно випущена у 2011 році. Ця бібліотека пропонує широкий спектр готових компонентів, таких як навігаційні панелі, кнопки, таблиці, форми та модальні вікна, що суттєво спрощує процес створення користувацьких інтерфейсів. Однією з ключових переваг Bootstrap є її простота у використанні та вбудована система адаптивної сітки (grid system) [2], яка забезпечує ефективну адаптацію інтерфейсу до різних типів пристроїв (рис. 1.3).

Рисунок 1.3: Логотип Bootstrap

Крім того, бібліотека сумісна з багатьма сучасними браузерами та має активну спільноту розробників, що сприяє регулярним оновленням і вдосконаленням. Завдяки високому рівню документації та наявності великої

кількості прикладів практичного застосування, Bootstrap є оптимальним вибором як для початківців, так і для досвідчених розробників. Зокрема, цю бібліотеку активно використовують на таких масштабних платформах, як Spotify та X (Twitter).

Tailwind CSS є утилітарною бібліотекою для стилізації, що дозволяє створювати складні користувацькі інтерфейси без необхідності написання власного CSS коду. Замість заздалегідь створених компонентів, вона пропонує набір класів, які забезпечують можливість швидкого формування індивідуальних елементів, адаптованих під конкретні вимоги проєкту [4].

Завдяки високій гнучкості та можливостям кастомізації, Tailwind CSS є ефективним рішенням для проєктів, де потрібен унікальний дизайн. Активна спільнота розробників постійно розширює функціонал бібліотеки та пропонує нові приклади її використання, що робить її відмінним вибором для застосунків з високими вимогами до адаптивності та продуктивності (рис. 1.4).

Рисунок 1.4: Логотип Tailwind CSS

Semantic UI - це бібліотека інтерфейсу користувача, яка базується на принципах семантичного HTML та CSS. Вона надає багатий набір компонентів та стилів, що дозволяють розробникам створювати зрозумілі та красиві інтерфейси. Semantic UI використовує людсько-читаний HTML, що спрощує код та робить його більш інтуїтивно зрозумілим (рис. 1.5).

Рисунок 1.5: Логотип Semantic UI

Semantic UI пропонує велику кількість налаштувань і підтримує адаптивний дизайн, що робить її популярним вибором серед розробників, які бажають створювати зрозумілі та привабливі інтерфейси.

Material-UI (MUI), Ant Design та Chackra є популярними бібліотеками компонентів, розробленими спеціально для використання з фреймворком React.

MUI є реалізацією принципів матеріального дизайну, розроблених Google. Бібліотека пропонує широкий набір компонентів, що відповідають рекомендаціям матеріального дизайну, спрямованим на досягнення гармонійної взаємодії, інтерактивності та зручності використання. До її складу входять такі елементи, як кнопки, картки, таблиці, списки, діаграми та навігаційні панелі (рис. 1.6).

Рисунок 1.6: Логотип Material-UI

Однією з основних переваг MUI є можливість повної кастомізації компонентів за допомогою стилів і тем, що дозволяє легко адаптувати інтерфейс до вимог конкретного продукту чи бренду. Така гнучкість робить бібліотеку популярним вибором серед розробників, які працюють з екосистемою React та прагнуть створювати функціональні й естетично привабливі додатки.

Ant Design - це UI бібліотека, розроблена компанією Alibaba для створення вебзастосунків на основі React. Вона надає великий набір високоякісних компонентів, що включають таблиці, форми, графіки, меню та інші елементи, орієнтовані на створення складних корпоративних додатків.

Ant Design має потужну систему тем, яка дозволяє легко змінювати вигляд і відчуття додатка.

Основними перевагами Ant Design є його комплексність і здатність задовольнити потреби великих проєктів. Бібліотека активно підтримується і розвивається, має детальну документацію та приклади використання, що робить її популярним вибором серед розробників корпоративного програмного забезпечення (рис. 1.7).

Рисунок 1.7: Логотип Ant Design

Chakra UI є сучасною бібліотекою інтерфейсу користувача для React, що вирізняється простотою, гнучкістю та модульною архітектурою. Вона надає широкий набір компонентів, оптимізованих для створення сучасних вебзастосунків, дозволяючи повністю кастомізувати стилі завдяки вбудованій системі тем і використанню сучасних технологій, таких як CSS-in-JS.

Однією з ключових переваг Chakra UI є висока доступність: усі компоненти спроєктовані з урахуванням стандартів ARIA [9, с.7]. ARIA - це специфікація, що покращує взаємодію з інтерфейсами для користувачів з особливими потребами, забезпечуючи інклюзивність вебзастосунків. Вона дозволяє розробляти елементи інтерфейсу, які доступні для користувачів із порушеннями зору, слуху або руховими обмеженнями, завдяки використанню додаткових атрибутів для покращення роботи з допоміжними технологіями, такими як екранні читачі. Chakra UI також забезпечує

гнучкість і масштабованість для різних проєктів, пропонуючи розробникам чудову підтримку та якісну документацію (рис. 1.8).

Рисунок 1.8: Логотип Chakra UI

Vuetify є однією з найпоширеніших бібліотек для фреймворку Vue.js, що реалізує принципи матеріального дизайну, розроблені Google. Ця бібліотека надає розробникам широкий набір компонентів для створення адаптивних, доступних та естетично привабливих інтерфейсів користувача. Використовуючи Vuetify, можна швидко інтегрувати сучасні дизайнерські рішення в вебзастосунки, що сприяє прискоренню процесу розробки та забезпечує стандартизований підхід до дизайну (рис. 1.9).

Рисунок 1.9: Логотип Vuetify

Гнучкість та підтримка кастомізації роблять Vuetify універсальним інструментом, що дозволяє налаштовувати компоненти відповідно до потреб конкретного проєкту. Завдяки активній підтримці спільноти та постійним оновленням, бібліотека зберігає сумісність із новими версіями Vue і дотримується найкращих практик веброзробки. Це забезпечує створення

адаптивних та інклюзивних інтерфейсів, що відповідають вимогам доступності та сучасним стандартам вебдизайну.

1.3. Переваги та недоліки використання UI бібліотек

UI бібліотеки відіграють важливу роль у сучасній розробці вебзастосунків і мобільних додатків, оскільки надають готові компоненти та рішення, що значно спрощують процес створення користувацьких інтерфейсів. Проте, як і будь-який інструмент, їх використання має як переваги, так і обмеження (табл. 1.3).

Таблиця 1.3.

Переваги та недоліки використання UI бібліотек

Переваги	Недоліки
Швидкість розробки за рахунок готових компонентів, що зменшує час на створення інтерфейсу з нуля.	Обмеження у гнучкості дизайну, що може вимагати додаткових ресурсів для адаптації під специфічні вимоги.
Узгодженість стилю інтерфейсу по всьому проекту завдяки стандартним компонентам.	Залежність від сторонніх розробників бібліотеки, що може створити ризики у разі припинення підтримки.
Підтримка принципів доступності (accessibility) для користувачів з особливими потребами.	Збільшення розміру додатку, що негативно впливає на швидкість завантаження.
Адаптивні компоненти, що автоматично підлаштовуються під різні розміри екранів.	Надмірність функцій та компонентів, які можуть бути непотрібними для конкретного проекту, ускладнюють код.
Регулярні оновлення та виправлення помилок завдяки активній спільноті.	Складність інтеграції з іншими інструментами, фреймворками або існуючим кодом, що вимагає

	додаткових налаштувань.

Продовження таблиці 1.3

Легка доступність навчальних матеріалів, документації та прикладів спрощує процес розвитку проекту.	Незважаючи на наявність документації, вивчення бібліотеки і освоєння її функцій займає час, що уповільнює початкові етапи розробки, особливо для нових учасників команди.
Оптимізація продуктивності у деяких бібліотеках для покращення користувацького досвіду.	Оновлення бібліотеки може викликати конфлікти або поломки в існуючому коді.

UI бібліотеки значно спрощують і пришвидшують розробку інтерфейсів, надаючи готові рішення, проте їх використання має свої обмеження. Вони сприяють покращенню ефективності командної роботи і підтримці узгодженого стилю, але можуть обмежувати гнучкість дизайну та викликати залежність від сторонніх розробників. Важливо зважено підходити до вибору UI бібліотеки, оцінюючи її відповідність конкретним завданням проекту, щоб уникнути можливих проблем з продуктивністю, підтримкою та інтеграцією.

1.4. Вплив UI бібліотек на користувацький досвід

У перші роки існування Інтернету більшість вебсайтів мали досить примітивний і неестетичний вигляд. Ці ранні сайти, створені в 1990-х роках, часто використовували простий текст, основні кольори і прості графічні елементи. Навігація по таких сайтах була неінтуїтивною, а інтерфейси - заплутаними і незручними для користувачів. Приклади таких вебсайтів, як ранні версії Yahoo! та Craigslist (сайт якого зберігся до сьогодні у своєму

оригінальному вигляді), демонструють відсутність структурованості та візуальної привабливості. На таких сайтах складно знайти потрібну інформацію, а час завантаження часто був значним через неефективні методи розробки.

Розуміння важливості користувацького досвіду (User Experience, UX) почало зростати на початку 2000-х, коли компанії усвідомили, що добре спроектовані сайти не тільки виглядають краще, але й сприяють підвищенню задоволеності користувачів і збільшенню конверсій. Піонерами в дослідженнях UX стали такі фахівці, як Дональд Норман, Джакоб Нільсен і Джефф Раскін.

Дональд Норман (Donald Norman), який ввів термін "User Experience" ще в 1990-х роках, відіграв важливу роль у розвитку розуміння важливості UX [10]. Норман підкреслює значення зручності використання та зрозумілості інтерфейсу для створення позитивного користувацького досвіду. Він стверджував, що гарний дизайн має бути орієнтований на користувача, а не на технологію.

Джакоб Нільсен, співзасновник Nielsen Norman Group та один із піонерів у сфері UX-досліджень, зазначає [11]: "Евристичний аналіз дозволяє виявити основні користувацькі проблеми на ранніх етапах розробки, що дає змогу підвищити ефективність і якість інтерфейсу без значних витрат часу і ресурсів". Він також пропонує методи, такі як когортний тестинг, що дозволяють швидко оцінювати якість інтерфейсу та поліпшувати його для забезпечення кращого користувацького досвіду.

Джефф Раскін, один із творців Macintosh у компанії Apple, підкреслював [12]: "Справді гуманний інтерфейс - це такий, що відповідає когнітивним потребам користувачів і мінімізує складність взаємодії". Він стверджував, що дизайн програмного забезпечення має бути інтуїтивно зрозумілим і

максимально спрощеним, забезпечуючи зручність і легкість використання для кінцевого користувача.

З початку 2000-х років такі компанії, як Google, Apple, Microsoft і Amazon почали активно інвестувати в UX-дослідження, розуміючи, що користувацький досвід є ключовим фактором успіху цифрових продуктів.

Google зіграла важливу роль у популяризації UX-досліджень, особливо з моменту свого заснування. Google Analytics, Google Search Console та інші інструменти дозволяють дослідникам вивчати поведінку користувачів на вебсайтах та збирати важливі дані для поліпшення інтерфейсів. Один з провідних фахівців Google у сфері UX, Матіас Дуарте, зазначив у своїй роботі [13]: "Material Design спрямований на створення візуально узгоджених та інтуїтивно зрозумілих інтерфейсів, які забезпечують єдиний досвід користувача незалежно від платформи чи пристрою". Він підкреслював важливість використання єдиних візуальних та функціональних принципів для покращення взаємодії з користувачем.

Джоні Айв, головний дизайнер Apple, у своїй роботі наголошував [14 с.32]: "Простота - це не лише відсутність безладу, це свідоме підкреслення найважливіших елементів, що забезпечують функціональність і естетичність продукту". Він активно розвивав UX-дослідження, впроваджуючи нові підходи до дизайну, що поєднують апаратне та програмне забезпечення, і підкреслював важливість гармонії між простотою та функціональністю як основи успіху продуктів Apple.

Microsoft довгий час зосереджувалася на розробці інтуїтивних інтерфейсів для своїх продуктів, таких як Windows і Office. В компанії функціонує окремий підрозділ, який займається виключно дослідженням UX та юзабіліті, впроваджуючи отримані результати в продукти компанії.

Amazon зі своєю фокусом на покращенні досвіду покупця використовує UX-дослідження для оптимізації процесів покупки, навігації та пошуку на

своєму сайті. Компанія також активно впроваджує A/B-тестування, аналізуючи поведінку користувачів, щоб знайти найефективніші рішення.

Сьогодні UX-дослідження є багатогранною дисципліною, яка включає різноманітні підходи для покращення взаємодії користувачів з продуктами. Вона охоплює аналіз поведінкових, когнітивних та емоційних аспектів взаємодії, спрямованих на оптимізацію інтерфейсів та підвищення загальної ефективності продуктів. Важливим аспектом UX-досліджень є використання різних методик, які дозволяють виявляти проблеми користувацького досвіду та знаходити способи їх вирішення.

Однією з поширених методик є A/B-тестування, яке використовується для порівняння двох версій вебсторінки або інтерфейсу. Цей метод дозволяє визначити, яка з версій забезпечує кращий користувацький досвід, спираючись на кількісні показники взаємодії. A/B-тестування дає змогу розробникам оптимізувати інтерфейси, покращуючи показники взаємодії користувачів і ефективність продуктів.

Іншим суттєвим методом є картування користувацьких подорожей, що аналізує всі точки контакту користувача з продуктом або сервісом. Цей підхід дозволяє виявляти критичні етапи взаємодії - від першого знайомства до післяпродажного обслуговування - і знаходити можливості для вдосконалення.

Також значущим елементом UX-досліджень є аналіз юзабіліті, який спрямований на оцінку зручності використання інтерфейсу. Цей метод допомагає визначити, наскільки інтерфейс відповідає вимогам навігації, зрозумілості та загальної взаємодії, виявляючи можливі бар'єри та підвищуючи загальний рівень користувацької зручності.

Сучасні UX-дослідження стали важливою складовою розробки програмних продуктів, орієнтованих на створення зручних, доступних і привабливих користувацьких інтерфейсів. Використання UI бібліотек у

цьому процесі відіграє значну роль, оскільки вони забезпечують розробників готовими компонентами, що пришвидшують процес створення інтерфейсу. Проте, як і будь-який інструмент, UI бібліотеки мають свої переваги та недоліки, що впливають на користувацький досвід (UX). Важливо оцінювати, наскільки відповідність цих бібліотек конкретним вимогам проєкту сприяє досягненню високоякісного користувацького досвіду.

- Однією з найбільших переваг UI бібліотек є забезпечення консистентності інтерфейсу по всьому проєкту. Всі компоненти дотримуються єдиного стилю та поведінки, що створює передбачуване та комфортне середовище для користувачів. Консистентність дизайну дозволяє зменшити когнітивне навантаження на користувачів, оскільки їм не доводиться вивчати нові способи взаємодії на різних етапах роботи з продуктом;

- UI бібліотеки зазвичай включають оптимізовані навігаційні компоненти, такі як меню, панелі управління та кнопки, що полегшує пошук потрібної інформації та виконання завдань. Добре спроектовані навігаційні елементи сприяють швидшій адаптації користувачів до інтерфейсу і підвищують загальну ефективність взаємодії;

- Сучасні UI бібліотеки часто відповідають стандартам вебдоступності, зокрема WCAG. Вони забезпечують підтримку користувачів із різними фізичними та когнітивними потребами, зокрема через вбудовані функції, як-от клавіатурна навігація або використання ARIA-атрибутів. Це підвищує доступність додатків і забезпечує інклюзивний користувацький досвід;

- Компоненти UI бібліотек зазвичай оптимізовані для різних типів пристроїв і розмірів екранів. Це дозволяє створювати респонсивні інтерфейси, що автоматично підлаштовуються під мобільні пристрої, планшети та десктопи. Адаптивний дизайн є критично важливим для

забезпечення комфортного досвіду взаємодії в умовах сучасного мобільного середовища;

- Багато UI бібліотек розробляються з урахуванням оптимізації продуктивності, що дозволяє зменшити час завантаження сторінок і підвищити загальну швидкість рендерингу інтерфейсу. Це особливо важливо для мобільних пристроїв, де пропускна здатність може бути обмеженою, і кожна секунда завантаження впливає на користувацький досвід;

- Використання стандартних компонентів і патернів дозволяє користувачам легше зрозуміти, як працювати з інтерфейсом, що знижує кількість помилок і підвищує зручність використання. Знайомі елементи дизайну полегшують процес навчання користувача, сприяючи швидшій адаптації до нового продукту.

Водночас, використання готових компонентів може обмежувати можливість створення унікальних і креативних інтерфейсів, які відображають специфічні вимоги бренду або користувачів. Стандартизовані рішення іноді не дозволяють досягти високого рівня індивідуалізації, що може знижувати конкурентоспроможність продукту на ринку:

- Деякі UI бібліотеки можуть включати безліч непотрібних компонентів, що збільшують обсяг проєкту та час завантаження сторінок. Це негативно впливає на продуктивність продукту, особливо в умовах слабких мережевих з'єднань або на мобільних пристроях із обмеженими ресурсами;

- Готові компоненти не завжди можуть повністю відповідати унікальним вимогам проєкту або очікуванням користувачів. Наприклад, певні функціональні можливості можуть бути відсутні або недостатньо гнучкі для кастомізації, що потребує додаткових зусиль для адаптації бібліотеки під конкретні потреби;

- Хоча більшість сучасних UI бібліотек враховують вимоги доступності, деякі складні інтерактивні елементи можуть не повністю підтримувати взаємодію з клавіатурою або екранними читачами. Це може створювати бар'єри для користувачів з особливими потребами, що негативно впливає на інклюзивність продукту;

- Для ефективного використання UI бібліотек розробникам може знадобитися додатковий час на навчання та освоєння їхньої архітектури. Це може сповільнювати процес розробки та створювати додаткові складнощі під час інтеграції бібліотеки в проєкт, особливо для нових членів команди або менш досвідчених розробників.

Висновки до розділу 1

UI бібліотеки є важливими інструментами в розробці сучасних вебзастосунків і мобільних додатків, пропонуючи готові компоненти для створення зручних та адаптивних інтерфейсів. Вони класифікуються за типами застосування, функціональністю та технологічними платформами, що дозволяє розробникам обирати оптимальні рішення відповідно до потреб проєкту. Серед популярних UI бібліотек можна виділити React, Bootstrap, Material-UI, що забезпечують широкий спектр функцій.

Використання UI бібліотек має численні переваги, такі як прискорення процесу розробки, поліпшення доступності та стандартизація інтерфейсів, проте існують і недоліки, включаючи обмеження у дизайні та можливі проблеми з інтеграцією. Загалом, UI бібліотеки позитивно впливають на користувацький досвід (UX), проте вибір відповідної бібліотеки повинен базуватися на ретельному аналізі специфіки проєкту та вимог до продуктивності.

РОЗДІЛ 2.
АНАЛІЗ ВПЛИВУ UI БІБЛІОТЕК НА ПРОЦЕС РОЗРОБКИ ДОДАТКІВ

2.1. Вплив UI бібліотек на швидкість та ефективність розробки

UI бібліотеки відіграють значну роль у прискоренні та оптимізації процесу розробки веб та мобільних застосунків. Вони надають готові рішення для створення інтерфейсів користувача, що дозволяє розробникам зосередитися на вирішенні більш специфічних завдань, водночас значно знижуючи час та ресурси, необхідні для розробки базових елементів.

Як вже було зазначено, UI бібліотеки суттєво скорочують час, необхідний для розробки інтерфейсів, надаючи готові компоненти, такі як кнопки, форми, таблиці та модальні вікна. Завдяки цьому розробники позбавляються необхідності створювати базові елементи з нуля, що дозволяє значно зекономити час і зусилля. Наприклад, використання бібліотек на кшталт Bootstrap надає можливість інтегрувати вже готові рішення, не витрачаючи додатковий час на детальне опрацювання їх дизайну та функціональності, що в результаті пришвидшує розробку проєктів. Джон МакРі зазначає [15], що використання готових компонентів UI-бібліотек може прискорити розробку в кілька разів, оскільки знімає необхідність у ручному кодуванні базових елементів.

Для підтвердження був проведений експеримент, у межах якого була розроблена сторінка авторизації (рис. 2.1) за двома підходами: перший із використанням фреймворку Bootstrap (Додаток А), а другий – виключно на основі чистих JavaScript, CSS та HTML без залучення сторонніх бібліотек (Додаток Б). Метою експерименту було порівняння часу, необхідного на

виконання ключових етапів розробки, таких як створення компонентів, дизайн, тестування та адаптація для пристроїв. У процесі експерименту відстежувалися всі етапи розробки, що дозволило об'єктивно оцінити вплив застосування UI бібліотеки на загальний час та ефективність розробки.

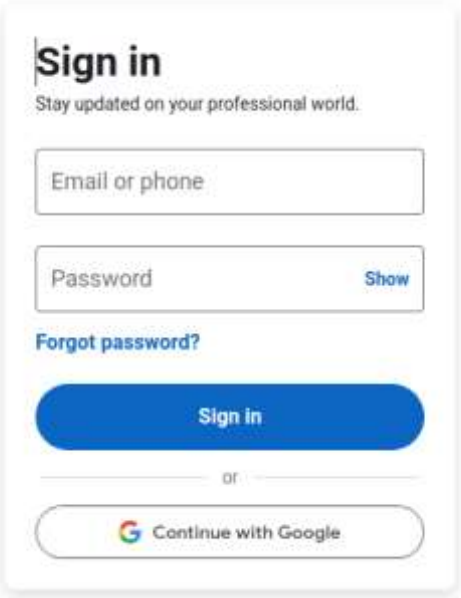


Рисунок 2.1: Сторінка авторизації

На основі отриманих результатів (рис. 2.2), було виявлено, що використання UI бібліотеки Bootstrap значно скоротило час розробки логін-сторінки в три рази у порівнянні зі створенням аналогічної сторінки без використання додаткових бібліотек. Загальний час розробки з використанням Bootstrap склав 5 годин, тоді як створення сторінки без UI бібліотеки вимагало 12 годин. Така різниця свідчить про значний вплив готових UI рішень на оптимізацію процесу розробки та підвищення його ефективності.

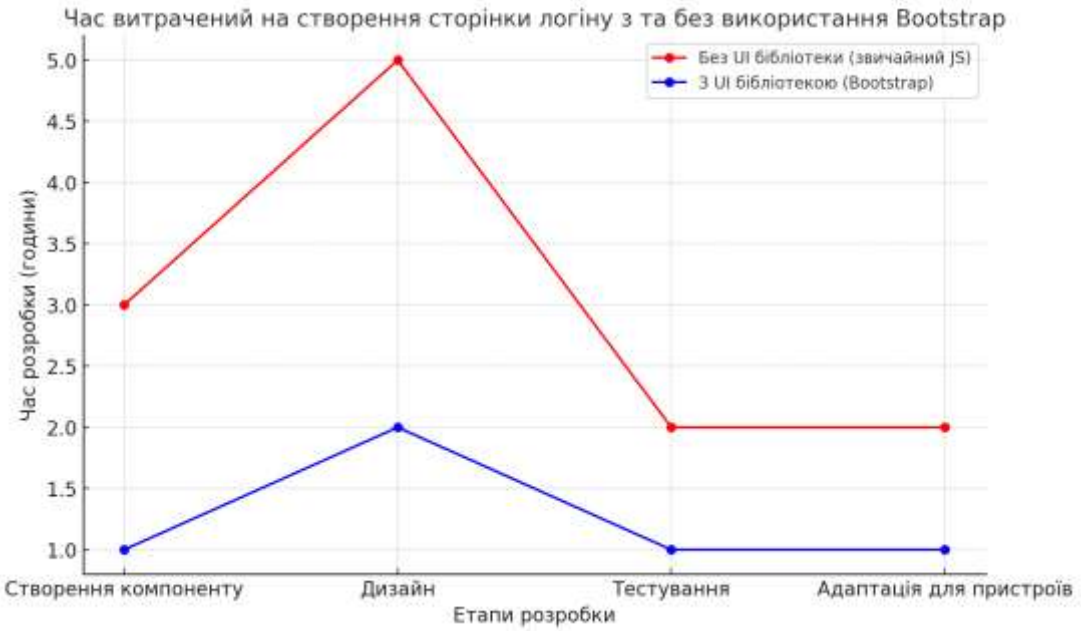


Рисунок 2.2: Графік часу розробки

Важливо зазначити, що пришвидшення процесу розробки за рахунок використання UI бібліотек, таких як Bootstrap, не є однаковим для всіх типів проєктів і значною мірою залежить від їх масштабу. У випадку невеликих чи середніх проєктів, таких як розробка сторінки авторизації, застосування бібліотек дозволяє значно оптимізувати робочі процеси, що підтверджується результатами експерименту. Проте, зі збільшенням масштабу проєкту виникає тенденція до уповільнення процесу розробки, що пов'язано з додатковими вимогами замовника щодо впровадження нових функцій та складніших компонентів. Це призводить до зростання обсягу робіт, необхідних для інтеграції нових елементів у вже існуючу структуру, а також потребує більше часу на тестування та адаптацію системи до зміни вимог.

Окрім значного прискорення процесу розробки, застосування UI бібліотек сприяє впровадженню єдиного стандарту дизайну та поведінки компонентів, що забезпечує послідовність інтерфейсу в межах проєкту. Це не тільки підвищує зручність і сприйняття користувачами, але й полегшує роботу розробників, які мають можливість використовувати стандартизовані

елементи та шаблони. Такий підхід дозволяє уникати потенційних конфліктів у дизайні та структурі інтерфейсу, що сприяє підвищенню якості кінцевого продукту.

При цьому важливу роль у забезпеченні високої якості відіграє тестування програмного забезпечення. Як підкреслює Роберт Мартін [16], тестування програмного забезпечення є невід'ємною складовою розробки додатків з кількох причин, які суттєво впливають на якість кінцевого продукту. Тестування дозволяє виявити дефекти на ранніх стадіях розробки.

Складність сучасних програмних продуктів залишає багато можливостей для помилок, тому тестування є критичним інструментом для їх виявлення та виправлення ще до випуску продукту. Це суттєво знижує ризики збоїв і несправностей на пізніших етапах. Тестування виявляє потенційні проблеми та гарантує, що кожен компонент працює відповідно до очікувань в різних умовах використання. Невідтестований продукт може призвести до значних витрат часу та ресурсів на виправлення помилок після його випуску. Якісне тестування компонентів забезпечує не лише технічну стабільність, але й загальний успіх проєкту, мінімізуючи негативні наслідки для користувачів.

Застосування модульної архітектури, властивої сучасним UI бібліотекам, спрощує процес заміни або оновлення окремих компонентів без порушення загальної структури додатку. Це значно спрощує підтримку та обслуговування проєкту, оскільки локальні зміни можуть бути здійснені без ризику виникнення проблем у функціонуванні інших елементів інтерфейсу. Модульність дозволяє більш гнучко реагувати на вимоги щодо розширення функціоналу або адаптації до нових потреб.

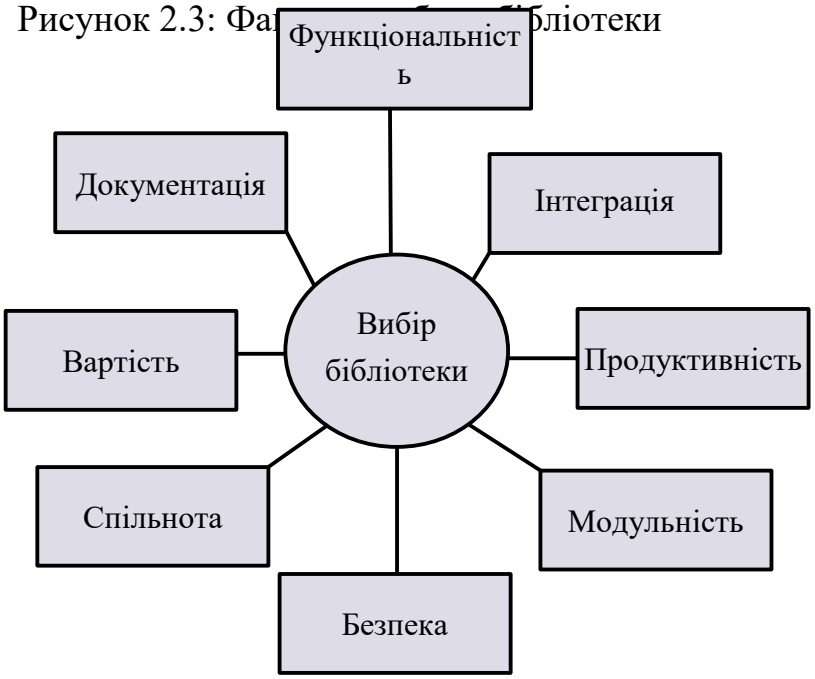
Крім того, UI бібліотеки регулярно оновлюються відповідно до нових технологічних стандартів, що дозволяє розробникам легко інтегрувати сучасні функції у свої проєкти. Наприклад, раніше зазначений Tailwind CSS

постійно оновлюється, забезпечуючи актуальні інструменти для адаптації дизайну відповідно до нових вимог ринку та користувачів. Це надає можливість розробникам швидко адаптувати свої проєкти до змін, що є важливим фактором у сучасних умовах динамічного розвитку технологій.

2.2. Аналіз факторів впливу на вибір бібліотеки

Вибір UI бібліотеки для розробки веб- або мобільного застосунку є критично важливим етапом, оскільки від цього рішення залежить не лише ефективність процесу розробки, але й якість та стійкість кінцевого продукту. У зв’язку з цим необхідно враховувати кілька ключових чинників, які визначають доцільність вибору тієї чи іншої бібліотеки (рис. 2.3).

Рисунок 2.3: Фа бліотеки



Перше, що слід оцінити при виборі бібліотеки, це її функціональність та можливості. Бібліотека повинна відповідати специфічним вимогам проекту і

забезпечувати необхідні компоненти та інструменти для реалізації бажаного інтерфейсу. Наприклад, якщо проект потребує складної інтерактивності та анімацій, бібліотеки, такі як Material-UI для React, можуть бути більш доречними ніж простіші рішення.

Якісна документація є одним із ключових критеріїв під час вибору UI бібліотеки для проєкту. Детальні описи API, приклади використання, а також навчальні матеріали значно полегшують процес освоєння бібліотеки розробниками, незалежно від їхнього рівня досвіду. Наявність такої документації скорочує час на інтеграцію бібліотеки в проєкт, дозволяючи розробникам швидше приступати до безпосереднього створення функціоналу. Документація також сприяє зниженню ймовірності виникнення помилок під час використання компонентів та інструментів бібліотеки, що позитивно впливає на загальну продуктивність команди розробки.

Крім того, наявність технічної підтримки відіграють важливу роль у процесі вибору бібліотеки. Велика кількість учасників спільноти забезпечує швидке розв'язання виникаючих питань та проблем, а також сприяє регулярному оновленню бібліотеки. Наявність форумів, відкритих репозиторіїв та технічних обговорень допомагає розробникам оперативно отримувати необхідну допомогу, зменшуючи ризики, пов'язані з використанням бібліотеки в критичних частинах проєкту. Це особливо важливо для великих проєктів, де своєчасне вирішення технічних проблем безпосередньо впливає на успішність і строки виконання завдань.

Сумісність бібліотеки з іншими технологіями, які використовуються в проєкті, є ще одним важливим аспектом вибору. UI бібліотека повинна легко інтегруватися з фреймворками, такими як React, Vue або Angular, а також з іншими інструментами розробки та системами управління контентом. Якщо проєкт уже побудований на певному фреймворку, важливо, щоб обрана бібліотека не лише підтримувала цей фреймворк, але й надавала засоби для

безшовної інтеграції, що забезпечує ефективність розробки. Наприклад, у разі використання фреймворку React доцільно обрати бібліотеку, яка легко працюватиме з компонентами React та забезпечуватиме мінімальні проблеми сумісності.

Продуктивність бібліотеки також є вирішальним фактором, особливо для проєктів, де важлива швидкодія та мінімальні затримки у відображенні інтерфейсу. Висока швидкість рендерингу компонентів, ефективна робота з DOM та загальне навантаження на клієнтську частину застосунку впливають на те, наскільки зручним і швидким буде досвід взаємодії користувачів із застосунком. Як зазначає Ерік Фрімен [17], бібліотеки з оптимізованим рендерингом, такі як React, забезпечують кращу продуктивність завдяки віртуальному DOM, що мінімізує кількість оновлень реального DOM. UI бібліотеки з подібними алгоритмами є кращим вибором для проєктів, які потребують високої продуктивності. Важливо, щоб бібліотека не суттєво уповільнювала завантаження сторінок або роботу самого застосунку, що є критичним для великих і навантажених систем.

Модульність бібліотеки визначає можливість використання лише тих компонентів, які потрібні для реалізації конкретних завдань, без перевантаження застосунку зайвими елементами. Бібліотеки, що підтримують гнучку налаштовуваність компонентів, як-от Tailwind CSS, дозволяють розробникам створювати індивідуальні інтерфейси відповідно до вимог проєкту, при цьому зберігаючи легкість і продуктивність застосунку. Це особливо важливо для великих проєктів, де надлишкові залежності можуть призводити до перевантаження системи та зниження продуктивності.

Ліцензійні умови є ще одним важливим критерієм при виборі UI бібліотеки. Необхідно ретельно вивчити ліцензію бібліотеки, особливо якщо планується її використання в комерційних проєктах. Деякі бібліотеки можуть мати обмеження на безкоштовне використання або вимагати придбання

платних ліцензій для доступу до розширених функцій і технічної підтримки. Тому важливо переконатися, що обрана бібліотека відповідає потребам проєкту та бюджету, щоб уникнути непередбачених витрат.

Останній, але не менш важливий аспект - безпека бібліотеки. Це є критичним фактором, особливо для веб-застосунків, які взаємодіють із чутливими даними. Важливо переконатися, що бібліотека регулярно оновлюється для виправлення вразливостей та відповідає сучасним стандартам безпеки [18]. Бібліотеки, які мають добру репутацію в галузі безпеки, дозволяють мінімізувати ризики появи вразливостей і захищають проєкт від потенційних атак. Відсутність регулярних оновлень або відомі проблеми з безпекою можуть значно підвищити ризик для проєкту, тому вибір на користь стабільних і перевірених бібліотек є надзвичайно важливим.

2.3. Вибір UI бібліотеки в залежності від типу проєкту

Вибір UI бібліотеки є ключовим рішенням, що безпосередньо впливає на ефективність розробки та якість кінцевого продукту. Кожен тип проєкту має унікальні вимоги до інтерфейсу користувача, тому вибір бібліотеки повинен відповідати специфічним потребам конкретного проєкту.

Для корпоративних веб-додатків, які зазвичай мають складні інтерфейси з численними компонентами і функціоналом, важливо вибирати бібліотеки, які забезпечують високу продуктивність і можливість гнучкого налаштування. Компонентні бібліотеки, такі як Material-UI або Ant Design, часто є кращими варіантами для таких проєктів. Вони пропонують широкий спектр готових компонентів і рішень для створення складних інтерфейсів з урахуванням корпоративних стандартів і вимог до юзабіліті. Для

корпоративних веб-додатків ключовими є можливості кастомізації та продуктивність, хоча можуть виникати складнощі з інтеграцією (табл. 2.1).

Таблиця 2.1

Переваги та недоліки корпоративно орієнтованих бібліотек

Переваги	Недоліки
Широкий набір компонентів і шаблонів для корпоративних інтерфейсів.	Складність в освоєнні та інтеграції.
Розширені можливості для кастомізації та налаштування.	Великий обсяг коду може впливати на швидкість завантаження
Підтримка крос-браузерної сумісності та високий рівень продуктивності.	

Для мобільних застосунків важливо вибирати бібліотеки, які оптимізовані для роботи на мобільних пристроях і забезпечують адаптивний дизайн. Бібліотеки, такі як React Native для застосунків на базі React, або Vuetify для Vue, які спеціально розроблені для мобільних платформ, є відмінним вибором [19, с.60]. Вони дозволяють створювати застосунки, які добре функціонують на різних пристроях. Для мобільних застосунків важлива адаптивність та можливість повторного використання коду, але кількість компонентів може бути обмеженою (табл. 2.2).

Таблиця 2.2

Переваги та недоліки мобільно орієнтованих бібліотек

Переваги	Недоліки
Адаптивність і оптимізація для мобільних пристроїв	Обмежене число компонентів порівняно з веб-бібліотеками

Продовження таблиці 2.2

Готові компоненти для створення мобільного інтерфейсу	Вимагатиме додаткового навчання для роботи з мобільними специфікаціями
Можливість використання єдиного коду для розробки на різних платформах	

Для інтернет-магазинів та eCommerce платформ важливі елементи, такі як адаптивний дизайн, інтерактивність та швидкість завантаження. Бібліотеки, які підтримують інтеграцію з платформами електронної комерції, такими як Bootstrap або Tailwind CSS, можуть бути корисними. Вони забезпечують необхідні компоненти для створення привабливих та функціональних інтерфейсів користувача. Для інтернет-магазинів критично важливі швидкість завантаження та адаптивність, але може знадобитися додаткове налаштування для специфічних функцій (табл. 2.3).

Таблиця 2.3

Переваги та недоліки бібліотек для eCommerce платформ

Переваги	Недоліки
Широкий вибір компонентів для eCommerce, таких як кнопки, форми, модальні вікна.	Необхідність додаткового налаштування для специфічних функцій eCommerce.
Адаптивний дизайн для різних пристроїв	Складність інтегрування з існуючими системами управління контентом.
Легкість у налаштуванні та швидкість розробки різних платформах	

Для блогових платформ та контентних веб-сайтів важливо мати бібліотеки, які забезпечують простоту у використанні та гнучкість у створенні контенту. JQuery та Bootstrap є добрим вибором для таких проектів, оскільки вони забезпечують простий у використанні синтаксис і готові компоненти для побудови чистого та функціонального інтерфейсу. Для блогових платформ перевагою є легкість у використанні та готові компоненти, проте може бути недостатньо складних функцій (табл. 2.4).

Таблиця 2.4

Переваги та недоліки бібліотек для блогових платформ

Переваги	Недоліки
Легкість у використанні та швидкість розробки	Обмежене число компонентів для більш складних функцій.
Готові компоненти для контентних веб-сайтів	Застарівання та втрата актуальності для сучасних веб-додатків.
Добре документовані і підтримуються великою спільнотою	

Для створення прототипів або MVP (Minimum Viable Product) важливо швидко розробити функціональність і отримати зворотний зв'язок. Бібліотеки, такі як Tailwind CSS, є гарним вибором для таких проєктів, оскільки вони дозволяють швидко створювати прості, але ефективні інтерфейси без необхідності складної настройки. Для прототипів або MVP швидкість розробки є основною перевагою, хоча додаткове налаштування може бути необхідним при переході до повноцінного продукту (табл. 2.5).

Таблиця 2.5

Переваги та недоліки бібліотек для прототипів та MVP

Переваги	Недоліки
Швидкість розробки прототипів та MVP.	Додаткове налаштування для переходу від прототипу до повноцінного продукту.
Простота у використанні та налаштуванні.Обмежені можливості для кастомізації в деяких бібліотеках.	Обмежені можливості для кастомізації в деяких бібліотеках.
Легкість в інтеграції з іншими інструментами та фреймворками.	

Вибір UI бібліотеки значно впливає на процес розробки та кінцевий результат проєкту. Корпоративні веб-додатки, мобільні застосунки, інтернет-магазини, блогові платформи та прототипи мають різні вимоги до інтерфейсу користувача, що вимагає обґрунтованого підходу до вибору інструментів. Залежно від специфіки проєкту, бібліотеки можуть забезпечувати гнучкість налаштування, високу продуктивність або швидкість розробки. Однак, слід враховувати потенційні обмеження для забезпечення оптимальної відповідності вимогам проєкту.

2.4. Методології тестування UI компонентів та бібліотек

Тестування UI компонентів і бібліотек є критично важливим етапом в процесі розробки, що забезпечує їх надійність, функціональність і відповідність вимогам користувачів. Ефективні методології тестування допомагають виявити помилки та недоліки, що можуть вплинути на загальний користувацький досвід та продуктивність програми. Існує кілька

основних видів тестування UI компонентів і бібліотек, кожен з яких має свої особливості та застосовується для досягнення різних цілей:

1. Юніт-тестування (Unit Testing) - це метод тестування програмного забезпечення, при якому перевіряється коректність роботи окремих компонентів або функцій UI в ізольованому середовищі. Основна мета юніт-тестування - виявити помилки на ранньому етапі розробки, забезпечуючи тим самим високу якість коду та його відповідність заданим вимогам.

Юніт-тестування забезпечує оперативне виявлення помилок та високе покриття коду, полегшуючи підтримку та розуміння проєкту, але може не враховувати взаємодію компонентів і вимагає значних ресурсів для комплексних систем (табл. 2.6).

Таблиця 2.6

Переваги та недоліки юніт-тестування

Переваги	Недоліки
Оперативне виявлення та виправлення помилок у конкретних компонентах, що дозволяє швидко локалізувати проблеми без необхідності перевірки всього проєкту.	Фокусується на перевірці окремих компонентів, не враховуючи їх взаємодію з іншими частинами системи. Помилки при інтеграції можуть залишитися непоміченими.
Кожен компонент або функція перевіряються на коректність виконання, що знижує ризик впливу невиявлених помилок на загальну функціональність системи.	Для комплексних компонентів може знадобитися написання великої кількості тестів, що вимагає значних зусиль і часу, особливо за наявності складних залежностей або бізнес-логіки.

Продовження таблиці 2.6

Тести служать додатковою документацією до коду, показуючи, як кожен компонент працює і які результати очікуються, що полегшує підтримку проєкту та інтеграцію нових розробників.	Велика кількість юніт-тестів не гарантує виявлення всіх помилок, особливо тих, що виникають при інтеграції компонентів або в реальних сценаріях використання.

2. Інтеграційне тестування є важливою частиною процесу забезпечення якості програмного забезпечення, яке фокусується на перевірці взаємодії між різними компонентами системи або між компонентами користувацького інтерфейсу (UI) і їхніми залежностями. Це тестування перевіряє, як окремі частини системи функціонують разом, коли їх об'єднують для досягнення спільної мети.

У ході інтеграційного тестування особлива увага приділяється коректності обміну даними між компонентами, відповідності викликів API та правильності їх взаємодії. Це включає перевірку коректності обміну даними, відповідності API і правильності взаємодії між компонентами.

Типи інтеграційного тестування:

- Тестування інтеграції компонентів перевіряє, як окремі UI компоненти, такі як кнопки, форми або модальні вікна, працюють разом. Це може включати перевірку, як компоненти обробляють події та оновлюють стан;

- Тестування інтеграції з зовнішніми системами перевіряє взаємодію UI компонентів з зовнішніми сервісами, такими як бази даних, API або сторонні бібліотеки;

- Тестування інтеграції в реальному середовищі перевіряє, як система працює в умовах, максимально наближених до реальних, включаючи навантаження і конфігурації середовища.

Приклади інструментів для інтеграційного тестування:

- Postman. Використовується для тестування API, що дозволяє перевіряти взаємодію між клієнтським інтерфейсом та сервером, включаючи перевірку правильності відповідей і обробки запитів;

- Selenium. Інструмент для автоматизованого тестування веб-додатків, який дозволяє перевіряти взаємодію між різними компонентами веб-інтерфейсу, такими як форми, кнопки та навігаційні елементи;

- Jest. Хоча Jest більше відомий як інструмент для юніт-тестування, він також підтримує інтеграційні тести для перевірки взаємодії між компонентами, особливо в середовищі React.

Інтеграційне тестування дозволяє виявляти проблеми взаємодії між компонентами і забезпечує впевненість у правильній роботі системи, але може бути складним у налаштуванні та вимагати значних ресурсів для проведення тестів і покриття всіх можливих сценаріїв (табл. 2.7).

Таблиця 2.7

Переваги та недоліки юніт-тестування

Переваги	Недоліки
Допомагає виявити проблеми, що виникають при об'єднанні різних частин системи, включаючи помилки взаємодії компонентів або проблеми з комунікацією між модулями.	Може бути складним в налаштуванні, оскільки потребує створення і підтримки середовища, яке точно відображає реальні умови використання.
Дозволяє перевірити, чи компоненти і модулі працюють разом відповідно до очікувань, що забезпечує впевненість у правильній інтеграції та безпомилковому виконанні функцій.	Вимагає більше ресурсів для написання та підтримки тестів, а також значних обчислювальних потужностей і зберігання даних для проведення тестів.
	Не забезпечує повної перевірки всіх можливих сценаріїв використання, оскільки тести можуть не охоплювати всі можливі комбінації інтеграцій

3. Функціональне тестування є критичним етапом у забезпеченні якості програмного забезпечення, фокусуючись на перевірці того, чи відповідає користувацький інтерфейс (UI) компонент або бібліотека специфікаціям і вимогам, зазначеним у документації. Це тестування перевіряє, чи всі функціональні можливості компонентів, такі як кнопки, форми, навігаційні елементи та інші інтерактивні елементи інтерфейсу, працюють правильно і відповідно до очікувань.

Функціональне тестування має на меті перевірити, чи відповідають UI компоненти своїм функціональним вимогам. Це включає перевірку того, як компоненти взаємодіють з користувачем і чи виконують вони свої функції відповідно до специфікацій.

Типи функціонального тестування:

- Тестування інтерактивних елементів перевіряє, чи правильно функціонують такі елементи, як кнопки, поля введення, переключатели і чекбокси. Це включає перевірку їхньої реакції на різні дії користувача;

- Тестування валідації даних перевіряє, чи компоненти коректно валідують введені дані, наприклад, чи перевіряють правильність формату електронної пошти або чи відображають відповідні повідомлення про помилки;

- Тестування навігації перевіряє, чи правильно працює навігація між різними частинами інтерфейсу, чи відповідають посилання і кнопки своїм призначенням.

Приклади інструментів для функціонального тестування:

- Cypress. Потужний інструмент для функціонального тестування, який забезпечує зручний інтерфейс для написання і виконання тестів, а також має вбудовану підтримку для відлагодження тестів в реальному часі;

- TestCafe. Забезпечує можливість автоматизованого функціонального тестування веб-додатків, підтримуючи сучасні браузери та можливість написання тестів на JavaScript або TypeScript.

Функціональне тестування перевіряє відповідність функціональним вимогам і допомагає виявити помилки, що впливають на користувацький досвід, але може не охоплювати всі сценарії використання і вимагає значних ресурсів для повного покриття тестів (табл. 2.8).

Таблиця 2.8

Переваги та недоліки функціонального тестування

Переваги	Недоліки
Перевіряє відповідність усім функціональним вимогам і специфікаціям, підтверджуючи, що компоненти чи бібліотека виконують заплановані функції та відповідають очікуванням користувачів.	Може не враховувати всі можливі сценарії використання та крайні випадки, що може призвести до пропуску деяких помилок або проблем.
Допомагає виявити помилки, які впливають на користувацький досвід, зокрема, некоректну поведінку елементів інтерфейсу або проблеми з обробкою даних, дозволяючи виправити їх до виходу продукту.	Забезпечення повного охоплення функціональності вимагає значних ресурсів і часу для написання та підтримки великої кількості тестових випадків.

4. Тестування юзабіліті є критичним аспектом забезпечення якості користувацького інтерфейсу, оскільки фокусується на перевірці того, наскільки зручний і зрозумілий інтерфейс для кінцевих користувачів. Мета цього тестування - забезпечити, що інтерфейс не тільки функціонує правильно, але й легко використовується і відповідає очікуванням реальних користувачів.

Тестування юзабіліті має на меті оцінити, наскільки ефективно і зручно користувачі можуть взаємодіяти з інтерфейсом. Це включає спостереження за тим, як користувачі виконують завдання, оцінку легкості навігації та загальної задоволеності від використання продукту. Наприклад, досліджується, чи легко користувачам знайти необхідні функції на сайті, чи зрозумілі їм інструкції та повідомлення, і чи вони швидко можуть виконати основні завдання.

Методи тестування:

- Сесії спостереження. Користувачі виконують завдання, і тестувальники спостерігають за їхньою взаємодією з інтерфейсом, щоб виявити будь-які проблеми або непорозуміння;

- Інтерв'ю. Після виконання завдань тестувальники проводять інтерв'ю з користувачами, щоб дізнатися про їхні відгуки та враження;

- Анкети та опитування. Використовуються для збору кількісних даних про досвід користувачів, таких як задоволеність, легкість використання та інші метрики.

Тестування юзабіліті надає цінну інформацію про реальний користувацький досвід, допомагаючи виявити приховані проблеми, але є затратним за часом і ресурсами та може призвести до суб'єктивних результатів, які потребують додаткового аналізу (табл. 2.9).

Таблиця 2.9

Переваги та недоліки тестування юзабіліті

Переваги	Недоліки
Надає цінну інформацію про реальний користувацький досвід, допомагаючи виявити реальні проблеми та труднощі, з якими стикаються користувачі, що дозволяє зробити інтерфейс більш інтуїтивним і зручним у використанні.	Затратне за часом і ресурсами, оскільки проведення сесій спостереження, інтерв'ю та аналіз зібраних даних вимагає значних зусиль з боку команди тестувальників.
Виявляє проблеми, які можуть залишатися непоміченими в інших типах тестування, таких як функціональне або інтеграційне тестування, зокрема незначні деталі дизайну або труднощі з навігацією.	Результати можуть бути суб'єктивними, оскільки враження і відгуки окремих користувачів можуть варіюватися, що ускладнює узагальнення висновків і вимагає додаткового аналізу для формування об'єктивної оцінки.

5. Тестування на відповідність фокусується на перевірці того, чи відповідає користувацький інтерфейс (UI) або бібліотека стандартам та нормативам, які регулюють створення веб-застосунків. Це включає перевірку відповідності до загальноприйнятих стандартів доступності, веб-стандартів і специфікацій, що визначають, як інтерфейс має виглядати та функціонувати для різних груп користувачів.

Метою тестування на відповідність є забезпечення того, щоб UI компоненти і бібліотеки відповідали встановленим стандартам та нормативам, що регулюють їх функціональність і доступність. Це включає перевірку відповідності до стандартів веб-доступності, таких як WCAG (Web Content Accessibility Guidelines), а також інших специфікацій, що впливають на загальний досвід користувача.

Тестування може охоплювати різні аспекти, такі як кольоровий контраст, доступність для користувачів з обмеженими можливостями, відповідність HTML і CSS стандартам, а також забезпечення, щоб інтерфейс коректно працював на різних пристроях і браузерах.

Методи тестування:

- Аудити доступності. Використовують інструменти та методики для перевірки відповідності стандартам доступності. Це може включати перевірку кольорового контрасту, тестування екранних зчитувачів та перевірку навігації клавіатурою;

- Перевірка відповідності стандартам веб-технологій. Оцінка того, чи відповідає код HTML, CSS та JavaScript сучасним стандартам, визначеним W3C (World Wide Web Consortium);

- Аналіз документації. Перевірка документації на предмет того, чи відповідає вона вимогам і рекомендаціям стандартів.

Тестування на відповідність гарантує дотримання стандартів і нормативів, забезпечуючи доступність для всіх користувачів, але вимагає спеціалізованих знань та інструментів і не завжди охоплює всі аспекти юзабіліті та функціональності (табл. 2.10).

Таблиця 2.10

Переваги та недоліки тестування на відповідність

Переваги	Недоліки
Гарантує відповідність UI компонентів стандартам і нормативам, що допомагає уникнути юридичних і фінансових наслідків при їх невиконанні.	Може вимагати спеціалізованих знань та інструментів, таких як сканери доступності та програмні засоби для аналізу коду, що підвищує складність і витрати на тестування.

Продовження таблиці 2.10

Критично важливе для забезпечення доступності інтерфейсу для всіх користувачів, включаючи людей з обмеженими можливостями, що робить веб-застосунки більш інклюзивними. дизайну або труднощі з навігацією.	Часто зосереджується на відповідності стандартам, не охоплюючи всі аспекти юзабіліті та функціональності, що може вплинути на загальний користувацький досвід.

6. Тестування продуктивності є критичним етапом оцінки якості користувацького інтерфейсу (UI), яке фокусується на перевірці швидкості та ефективності, з якою UI компоненти реагують на дії користувача і обробляють дані. Це тестування дозволяє визначити, наскільки швидко інтерфейс завантажується, як швидко він реагує на взаємодії користувачів, і як стабільно він працює під різними навантаженнями.

Тестування продуктивності має на меті оцінити швидкість завантаження компонентів, реактивність інтерфейсу на дії користувачів та стабільність під високими навантаженнями, щоб забезпечити оптимальну продуктивність і стабільність системи:

-Перевірка часу, необхідного для завантаження UI компонентів та сторінок веб-застосунку;

- Оцінка часу відповіді інтерфейсу на дії користувачів, такі як натискання кнопок, введення тексту або прокрутка.

Методи тестування:

- Стрес-тестування. Перевірка UI під максимальними можливими навантаженнями, щоб визначити межі продуктивності та виявити можливі проблеми, що виникають при перевищенні цих меж.

- Тестування швидкості. Вимірювання часу завантаження і відповіді інтерфейсу на дії користувачів. Це може включати використання

інструментів для моніторингу часу завантаження сторінок і швидкості виконання запитів.

Тестування продуктивності виявляє проблеми, що впливають на швидкість і користувацький досвід, допомагаючи оптимізувати роботу інтерфейсу, але вимагає використання спеціальних інструментів та значних ресурсів при тестуванні великих обсягів даних і високих навантажень (табл. 2.11).

Таблиця 2.11

Переваги та недоліки тестування на відповідність

Переваги	Недоліки
Виявляє проблеми, пов'язані зі швидкістю та продуктивністю, що можуть уповільнювати роботу інтерфейсу або негативно впливати на користувацький досвід, дозволяючи виправити їх до випуску продукту.	Вимагає використання спеціальних інструментів для вимірювання продуктивності (JMeter, LoadRunner тощо), що може збільшити витрати на тестування.
Допомагає забезпечити оптимальний користувацький досвід, оцінюючи швидкість і стабільність інтерфейсу під різними навантаженнями, що сприяє утриманню користувачів.	Тестування може бути складним при великих обсягах даних і високих навантаженнях, потребуючи значних ресурсів і часу для налаштування та проведення тестів.

Висока якість користувацького інтерфейсу (UI) є критично важливою для успіху будь-якого веб-додатку чи мобільного застосунку. Вона не тільки впливає на функціональність продукту, але й на загальний досвід користувачів, який є ключовим чинником для їхнього задоволення та лояльності. Забезпечення якості інтерфейсу вимагає всебічного підходу до тестування, що включає різні методи перевірки, кожен з яких грає важливу роль у виявленні та усуненні проблем (табл. 2.12).

Таблиця 2.12

Порівняння методів тестування

Метод	Переваги	Недоліки	Рекомендоване використання
Юніт-тестування	- Оперативне виявлення помилок; - Високе покриття коду	- Не враховує взаємодію компонентів; - Потребує значних ресурсів для комплексних систем	Для швидкої перевірки окремих компонентів або функцій
Інтеграційне тестування	- Виявлення проблем у взаємодії між компонентами; - Перевірка коректності роботи в реальних умовах	- Складність налаштування і підтримки середовища; - Потребує значних ресурсів.	Для перевірки взаємодії компонентів, роботи з API чи базами
Функціональне тестування	- Підтвердження, що компоненти виконують свої функції; - Виявлення помилок, що впливають на користувачів	- Вимагає багато часу для повного покриття; - Не охоплює всі сценарії	Для перевірки основної функціональності та користувацьких вимог
Тестування юзабіліті	- Виявлення проблем, що можуть впливати на реальний досвід користувачів	- Суб'єктивність результатів; - Затратне за часом і ресурсами	Для оцінки користувацького досвіду і юзабіліті інтерфейсу
Тестування на відповідність	- Забезпечення доступності для всіх користувачів; - Відповідність стандартам	- Потребує спеціалізованих інструментів та знань	Для забезпечення відповідності стандартам і нормативам
Тестування продуктивності	- Виявлення проблем, що впливають на швидкість і стабільність інтерфейсу; - Оптимізація користувацького досвіду	- Потребує спеціальних інструментів та ресурсів для тестування великих обсягів даних	Для перевірки продуктивності та стабільності системи

Висновки до розділу 2

Аналіз впливу UI бібліотек на розробку програмного забезпечення та вебзастосунків демонструє, що вони суттєво прискорюють процес завдяки готовим компонентам і шаблонам. Це дозволяє розробникам зосередитися на бізнес-логіці та функціональності, а не на створенні інтерфейсів з нуля. Проте вибір бібліотеки має бути обґрунтованим, оскільки різні контексти вимагають різного підходу.

На вибір UI бібліотек впливають критичні фактори, такі як сумісність з технологічним стеком, підтримка документації, продуктивність і можливості кастомізації. Легкі бібліотеки можуть підходити для простих проєктів, тоді як для складних рішень потрібні потужніші варіанти. Тестування компонентів є важливим для забезпечення якості кінцевого продукту, адже систематичний підхід до тестування підвищує стабільність і надійність.

UI бібліотеки є важливими інструментами в сучасній розробці, адже вони підвищують швидкість і ефективність. Ретельний вибір, врахування специфіки проєкту та впровадження тестувань є ключовими для створення якісних і функціональних інтерфейсів користувача.

РОЗДІЛ 3.
РОЗРОБКА ВЛАСНОЇ UI БІБЛІОТЕКИ ДЛЯ ВЕБСАЙТІВ

3.1. Обґрунтування вибору технологій та інструментів для розробки

Розробка власної UI бібліотеки розпочалась через виявлену потребу на ринку, де бракувало сучасних рішень, здатних забезпечити максимальну функціональність без зайвого навантаження на ресурси системи. Під час аналізу архітектури, компонентів, адаптивності та продуктивності таких бібліотек, як Bootstrap, JQuery, Material-UI та інших, було помічено, що багато з них зосереджені на візуальних і стильових аспектах, включаючи складні анімації, теми та стилі. Це призводило до надмірного споживання ресурсів і зниження продуктивності, що погіршувало користувацький досвід через уповільнення завантаження сторінок.

З огляду на ці недоліки, було вирішено створити власне рішення, яке б мінімізувало додаткове навантаження на систему. Процес розробки складався з кількох етапів:

- планування архітектури бібліотеки, визначення ключових компонентів;

- реалізація за допомогою сучасних методів програмування мовою TypeScript із застосуванням компонентної архітектури;

- тестування та валідація для забезпечення сумісності та стабільної роботи на різних пристроях і браузерах;

- оцінка ефективності запропонованого рішення шляхом тестування та порівняння з існуючими UI бібліотеками.

Проєкт, що розробляється в рамках цього дослідження, отримав назву "Exadel Smart Library" (ESL) і був створений у співпраці з розробниками

Адамскою Ю.С., Стефановичем А.А., Лесун А.І. та Барміною А.А. які є співробітниками компанії Exadel.

Основна мета проєкту - створення легкої та швидкої UI бібліотеки, яка забезпечує лише необхідну функціональність вебсайтів, без надмірностей у вигляді важких фреймворків, таких як React. Цей підхід спрямований на оптимізацію роботи бібліотеки, зменшення її розміру та досягнення максимальної продуктивності при використанні в проєктах, де швидкість завантаження і рендерингу відіграє ключову роль.

Корпоративні вебсайти зазвичай включають велику кількість інтерактивних елементів, таких як форми, таблиці, динамічні віджети, панелі навігації тощо. Використання великих і важких бібліотек значно збільшує час завантаження таких елементів, що є критичним недоліком у світі, де користувачі очікують миттєвої реакції вебсайтів. Саме тому розробка власної бібліотеки дозволяє адаптувати її під специфічні потреби таких сайтів, надаючи можливість точкового контролю над усіма елементами інтерфейсу.

Додатковою перевагою власної розробки є можливість інтеграції бібліотеки з існуючими системами без конфліктів із стилями або функціональністю інших компонентів. Також важливим аспектом є зручність у підтримці та розширенні бібліотеки в майбутньому, адже її структура буде повністю підконтрольна розробникам, що спрощує внесення змін або додавання нових компонентів у відповідь на змінні вимоги корпоративних замовників.

Таким чином, розробка власної бібліотеки має мету забезпечити наступні аспекти (табл. 3.1).

Таблиця 3.1

Основні аспекти розробки власної бібліотеки:

Найменування	Мета
Оптиізація	Мінімізація кількості зайвих стилів та скриптів, що дозволяє прискорити завантаження сторінок
Гнучкість	Можливість налаштування під конкретні потреби корпоративного вебсайту без надмірного коду
Легкість підтримки	Власна бібліотека може бути легко модифікована та розширена відповідно до нових вимог
Інтеграція без конфліктів	Відсутність залежності від важких фреймворків забезпечує легку інтеграцію в будь-яке середовище.

Таким чином, UI бібліотека ESL має на меті забезпечити високий рівень функціональності при мінімальному навантаженні на систему. Її головними характеристиками є легкість і продуктивність, що досягається за рахунок відсутності вбудованих стилів та анімацій. Завдяки цьому бібліотека залишається гнучкою та легкою у налаштуванні, дозволяючи розробникам інтегрувати її в різноманітні проєкти без зайвого коду та залежностей. Відсутність зайвих стилів забезпечує чистий, оптимізований код, що ідеально підходить для корпоративних вебсайтів, де критично важливими є швидкість завантаження та ефективність рендерингу.

Для розробки власної UI-бібліотеки було обрано систему контролю версій Git, оскільки вона є надійним і перевіреним інструментом для керування змінами в коді. Система контролю версій (СКВ) - це інструмент для збереження і відстеження змін у програмному коді, що дозволяє

розробникам співпрацювати та керувати різними версіями проєкту. Історія систем контролю версій почалася ще у 1970-х роках, коли з'явилися перші інструменти для відстеження змін у вихідному коді, такі як SCCS та пізніший CVS.

Git, створений Лінусом Торвальдсом у 2005 році, є розподіленою системою контролю версій, яка надає розробникам можливість працювати незалежно, синхронізуючи зміни через розподілені репозиторії. Як зазначає Скотт Шакон [20], "Git надає можливість не тільки легко відслідковувати зміни, але й працювати одночасно кільком розробникам, синхронізуючи їхню роботу за допомогою розподілених репозиторіїв". Це дозволяє ефективно керувати проєктами, зокрема бібліотеками, де кожна зміна впливає на загальну функціональність.

Ще однією перевагою Git є інтеграція з різними хостинговими платформами, такими як GitHub, що надає зручні інструменти для керування проєктом, включаючи автоматизоване тестування, CI/CD (безперервна інтеграція та доставка), а також можливість відслідковування завдань і обговорення змін. Крім того, використання GitHub для розгортання бібліотеки через npm забезпечує швидкий доступ до кінцевого продукту для користувачів, що є важливим аспектом сучасного процесу розробки.

В рамках створення власної UI-бібліотеки ESL було обрано платформу NPM для її розгортання та розповсюдження. NPM (Node Package Manager) - це найбільший та найпопулярніший реєстр JavaScript-пакетів, який відіграє ключову роль в екосистемі Node.js. Він використовується для публікації, розповсюдження та управління пакетами, забезпечуючи розробникам можливість легко обмінюватися бібліотеками та інструментами.

Завдяки цьому, NPM надає швидкий доступ до вже готових рішень, що значно спрощує процес розробки і прискорює інтеграцію нових інструментів у проєкти. Цей вибір дозволив зробити бібліотеку доступною для широкої

спільноти розробників, надаючи простий і ефективний спосіб її інтеграції в різні проєкти. Використання NPM також полегшує процес оновлення та підтримки бібліотеки, спрощуючи випуск нових версій. Крім того, розміщення бібліотеки в NPM забезпечує зручне управління залежностями та контроль версій, що дозволяє встановлювати компоненти через стандартні механізми Node.js і покращує загальну стабільність та ефективність роботи з бібліотекою.

TypeScript було обрано як основну мову програмування для розробки бібліотеки, оскільки це надбудова над JavaScript, яка додає опціональні типи та високий рівень структурування коду, не навантажуючи при цьому вихідний код. Він компілюється у звичайний JavaScript, що дозволяє зберегти продуктивність, але водночас зменшує кількість помилок і підвищує надійність коду. Структурованість, яку забезпечує TypeScript, спрощує обслуговування та подальше розширення проєкту [21].

Webpack, інструмент для збірки модулів, також став ключовим компонентом під час розробки бібліотеки. Webpack дозволяє збирати всі ресурси проєкту (JavaScript, CSS, зображення тощо) в один або декілька файлів, оптимізуючи їх для виробничого використання. Цей інструмент особливо корисний для забезпечення високої продуктивності та мінімізації розміру файлів, що завантажуються браузером користувача. Завдяки можливостям модульної збірки, Webpack дозволяє зібрати всі залежності бібліотеки та впорядкувати їх так, щоб забезпечити максимальну ефективність під час завантаження вебсайтів [22]. Це сприяє прискоренню завантаження сторінок і поліпшенню загального користувацького досвіду.

Для забезпечення високої якості коду в проєкт було встановлено ESLint. Це інструмент для аналізу статичного коду, що допомагає виявляти та виправляти помилки. ESLint дозволяє встановити стандарти кодування, що зменшує ймовірність появи помилок і покращує читабельність коду. Це

важливо для підтримки стабільності проєкту, особливо в довгостроковій перспективі, коли над кодом можуть працювати кілька розробників. Завдяки ESLint проєкт відповідає сучасним стандартам і кращим практикам написання JavaScript/TypeScript, що сприяє більшій надійності та передбачуваності коду.

У проєкті для юнит-тестування було використано Jest - популярний інструмент для тестування JavaScript-коду. Юніт-тестування полягає в перевірці окремих модулів або компонентів програми на коректність їхньої роботи. Це дозволяє ізолювати кожен елемент системи та переконатися, що він функціонує належним чином незалежно від інших частин коду. Використання Jest у поєднанні з TypeScript дозволяє виявляти можливі помилки на ранніх етапах розробки та забезпечує стабільність програмного забезпечення.

Окрім юніт-тестування, для end-to-end (e2e) тестування в проєкт було додано Puppeteer - інструмент, який дозволяє автоматизувати браузерні дії для повного тестування взаємодії з додатком. e2e-тестування допомагає перевірити всю програму від початку до кінця, імітуючи поведінку реального користувача. У рамках цього процесу використовувалися снепшоти - це знімки поточного стану інтерфейсу або результатів виконання тесту, які зберігаються і порівнюються з майбутніми тестами. Якщо інтерфейс або результат змінюється, система попереджає про це, допомагаючи виявляти неочікувані зміни в додатку.

У проєкті для керування стильовими аспектами було обрано Less - потужний препроцесор CSS, який надає додаткові можливості для роботи зі стилями. Less дозволяє використовувати змінні, вкладені правила, міксіни та інші корисні функції, що робить процес написання CSS більш гнучким і зручним. Завдяки цьому розробники можуть легко керувати складними стилями, повторно використовувати код і підтримувати його в чистому та

організованому вигляді. Використання Less значно прискорює і спрощує роботу зі стилями, зберігаючи при цьому продуктивність і передбачуваність у проєкті.

3.2. Розробка та опис основних компонентів бібліотеки

Першим кроком у розробці проекту було створення репозиторію на платформі GitHub (https://github.com/exadel-inc/esl), що забезпечило зручне середовище для управління кодом. Це рішення дало можливість одразу організувати процес розробки, налаштувати автоматичне тестування та впровадити інструменти для забезпечення якості коду.

Структурування репозиторію стало важливим етапом, оскільки воно відіграє ключову роль у забезпеченні ефективності як розробки, так і підтримки проекту. Оптимальна організація дозволяє не лише спростити доступ до ресурсів проекту, але й чітко розподілити обов'язки між учасниками команди. Це сприяє полегшенню процесів версіонування та тестування, зменшуючи можливі конфлікти при роботі з кодом (рис. 3.1).

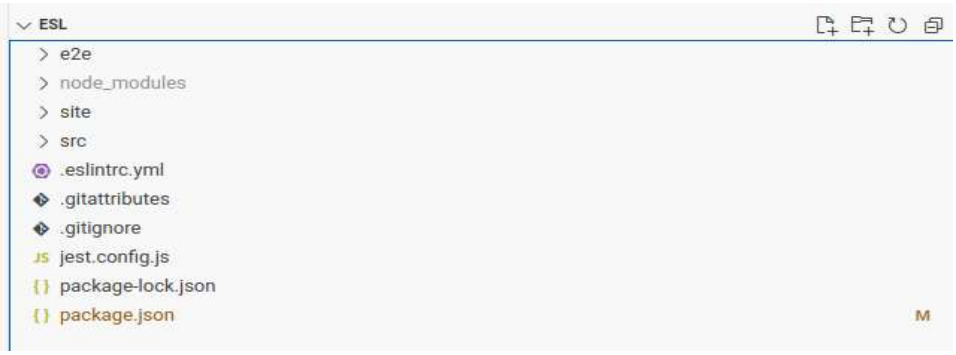


Рисунок 3.1: Загальна структура бібліотеки

Для опису структури репозиторію проекту, кожна директорія має свою специфічну роль і призначення, що допомагає впорядкувати процес розробки, тестування та демонстрації можливостей бібліотеки. З огляду на (рис. 3.1) можна виокремити наступні частини:

1. Директорія /sites. Ця директорія призначена для розміщення демонстраційних сторінок, які служать вітринам и можливостей кожного компоненту бібліотеки. Розробка демонстраційних сторінок є важливим етапом, оскільки вони не лише демонструють функціональність, але й служать платформою для проведення реальних користувацьких тестів. Вони виконують роль основного інтерфейсу для взаємодії розробників із створеними компонентами, забезпечуючи наглядне зображення внесених змін та нововведень.

2. Директорія /src/modules. Ця частина репозиторію містить код самої бібліотеки, розділений за модулями. Кожен компонент бібліотеки має свою індивідуальну піддиректорію, що спрощує процес інтеграції та подальшого розширення бібліотеки. Використання модульної структури сприяє ізоляції компонентів, що є особливо важливим для забезпечення незалежності та зменшення залежностей між різними частинами коду:

- /core - основний код кожного компоненту, включаючи логіку та стилізацію, знаходиться тут. Такий підхід дозволяє зосередити всі основні функціональні аспекти компоненту в одному місці, що значно спрощує процес розробки та подальшого рефакторингу.

- /tests - каталог для юніт тестів кожного з компонентів. Структурування тестів в окремій папці дозволяє системно підходити до верифікації функціональності, забезпечує легке управління тестовими сценаріями та їх налагодження.

3. Директорія /e2e. Цей каталог призначений для розміщення end-to-end (e2e) тестів та снепшотів, які використовуються для перевірки інтеграції

компонентів бібліотеки в реальних умовах користування. E2e тестування є ключовим для виявлення помилок, які можуть не бути виявлені під час юніт або модульного тестування. Ці тести симулюють дії користувачів, перевіряючи всю систему від початку до кінця, включаючи взаємодію з базою даних, сервером, і інтерфейсом користувача. Снепшоти забезпечують візуальну верифікацію того, що інтерфейс відповідає очікуваним стандартам та не зазнав несподіваних змін після оновлень коду.

4. Директорія /node_modules. Ця директорія генерується автоматично при встановленні залежностей через npm (Node Package Manager) і містить всі зовнішні бібліотеки та модулі, які використовуються в проекті. Папка /node_modules є фундаментальною для Node.js проектів, оскільки забезпечує ізоляцію залежностей кожного проекту, дозволяючи мати різні версії одних і тих же модулів для різних проектів без конфліктів. Включення цієї папки в структуру проекту сприяє ефективному управлінню залежностями та забезпечує стабільність і сумісність в робочому середовищі розробника.

Структура репозиторію має на меті досягнення двох основних цілей: відокремлення залежностей та забезпечення інтуїтивно зрозумілого пошуку коду.

У процесі розробки UI бібліотеки особлива увага була приділена створенню логотипу (рис. 3.2), який служить не лише як візуальний ідентифікатор, але й відображає філософію та ключові характеристики проекту. Логотип став важливим елементом брендингу, що забезпечує впізнаваність бібліотеки в середовищі розробників та користувачів.

Рисунок 3.2: Логотип ESL

В основу архітектури бібліотеки ESL було покладено базовий елемент - ESLBaseElement (Додаток В), який використовує можливості CustomElementRegistry для створення користувацьких веб-компонентів. Розглянемо детальні ше технологічний підхід та його реалізацію.

CustomElementRegistry є механізмом у веб API, який дозволяє розробникам визначати власні елементи та реєструвати їх в браузері. Цей реєстр дозволяє використовувати повністю налаштовувані теги разом з стандартними HTML-елементами, забезпечуючи інкапсуляцію стилів і поведінки. Компоненти, зареєстровані через CustomElementRegistry, мають змогу використовувати життєвий цикл веб-компонентів, що включає керування підключенням до дерева DOM, відслідковування змін атрибутів, та очищення після видалення з DOM.

ESLBaseElement є базовим класом для декларації користувацьких елементів. Цей клас надає фундаментальні API та утиліти для спрощення створення нових компонентів, використовуючи TypeScript декоратори для полегшення розробки.

API (Application Programming Interface) - це набір правил та специфікацій, які дозволяють програмам взаємодіяти одна з одною. В контексті веб-компонентів, API включає методи, властивості та події, які можуть бути використані для керування поведінкою компонентів та їхньою взаємодією з іншими елементами або додатками.

У бібліотеці ESL, API розділяється на статичний та екземплярний (інстанс) API веб-компонентів.

Статичний API стосується методів і властивостей, які доступні на рівні класу компонента. Це означає, що ці методи або властивості можуть бути викликані без необхідності створювати конкретний компонент на сторінці. Вони надають загальну інформацію або функціональність для всіх примірників компонента:

- ESLBaseElement.is - властивість, що визначає назву тегу компоненту;

- ESLBaseElement.observedAttributes - масив атрибутів, за якими компонент буде слідкувати.

Екземплярний API стосується методів і властивостей, які працюють з конкретними примірниками компонентів, створеними на сторінці. Ці методи використовуються для взаємодії з конкретним елементом після його ініціалізації в DOM. Екземплярний API дозволяє керувати поведінкою окремих елементів, реагувати на події та змінювати їхні атрибути:

- connectedCallback - викликається, коли елемент додається до DOM;

- disconnectedCallback - викликається, коли елемент видаляється з DOM;

- attributeChangedCallback - викликається при зміні спостережуваних атрибутів;

- $$cls - перевіряє або змінює CSS класи елемента;

- $$attr - перевіряє або змінює атрибути елемента;

- $$fire - ініціює подію;

- $$on - дозволяє підписатися на події вручну або за допомогою декорованого методу;

- $$off - відписує від події вручну або від декорованого методу.

Також API ESL реалізовані сучасні програмні рішення, зокрема декоратори, які спрощують та оптимізують взаємодію з компонентами. Декоратори у програмуванні - це спеціальні декларації, які можуть бути прикріплені до класу, методу або властивості. Вони забезпечують зручний спосіб модифікації поведінки класів або їх членів за допомогою зовнішнього коду. У контексті TypeScript та веб-компонентів, декоратори часто використовуються для налаштування метаданих компонентів, визначення властивостей атрибутів, та автоматичного реєстрування подій. В основу базового елементу ESLBaseElement було покладено наступні декоратори:

- @attr - відображає властивість типу рядок на HTML атрибут;

- @boolAttr - відображає булеву властивість на булевий (маркерний) стан атрибута;

- @jsonAttr - відображає властивість об'єкта на HTML атрибут, використовуючи JSON формат для серіалізації/десеріалізації значення;

- @listen - декорує метод з властивостями вбудованого JavaScript прослуховувача;

- @prop - декоратор для створення визначення значення на рівні прототипу.

На базі класу ESLBaseElement було створено ряд інших елементів, які використовуються для побудови інтерактивних інтерфейсів. Ці елементи надають розширені можливості для розробки компонентів з керуванням станом та взаємодією з користувачами:

1. Першим користувацьким елементом було створено ESLToggleable (Додаток Г), який є основою для розробки компонентів типу "спливаючих вікон" або інших елементів, що потребують активації/деактивації через користувацьку взаємодію (рис. 3.3).

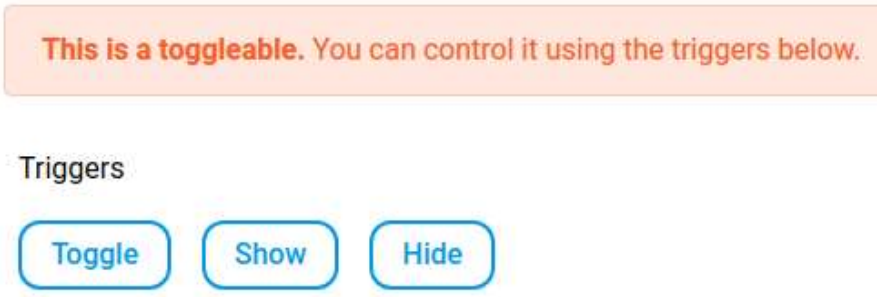


Рисунок 3.3: Елемент ESLToggleable

ESLToggleable є одним з компонентів бібліотеки, який надає зручний інтерфейс для роботи з елементами, що мають стан включення/вимкнення. Цей компонент є дуже гнучким та може бути використаний для створення таких елементів, як модальні вікна, спливаючі підказки або будь-які інші елементи, що мають бути видимі або приховані в залежності від дій користувача.

Компонент ESLToggleable базується на принципах інкапсуляції та реєстрації користувацьких елементів через механізм CustomElementRegistry, аналогічно до ESLBaseElement, що забезпечує інтеграцію з DOM через життєвий цикл веб-компонентів. Це дозволяє компоненту управляти своїм станом на рівні DOM та реагувати на зміни атрибутів чи взаємодії з іншими компонентами.

Компонент ESLToggleable реалізує публічний API для активації та деактивації елементів, що дає можливість контролювати їх стан через програмний інтерфейс. API включає наступні методи:

- show – викликає активацію елемента, роблячи його видимим чи активним;

- hide – деактивує елемент, приховуючи його або вимикаючи;

- toggle – перемикає стан елемента між активним і неактивним.

Це дозволяє розробникам легко інтегрувати такі елементи в свої проєкти, надаючи зручний спосіб керувати їх станом через стандартні методи.

Компонент ESLToggleable успадковує всі ключові можливості, надані базовим класом ESLBaseElement. Це означає, що ESLToggleable використовує всі API базового елемента, такі як connectedCallback, disconnectedCallback, attributeChangedCallback, а також підтримує роботу з атрибутами через такі утиліти, як $$attr, $$cls, $$fire. Це дозволяє легко

додавати та змінювати функціонал компонента, а також робить його інтеграцію з іншими компонентами зручнішою та ефективнішою.

Декоратори, що використовуються в ESLBaseElement, також можуть застосовуватися для розширення функціоналу ESLToggleable. Наприклад, можна використовувати декоратори для автоматичного відображення властивостей типу рядок або булевого стану в атрибути HTML. Це робить створення налаштовуваних компонентів максимально гнучким та зручним для розробників, зберігаючи при цьому високу продуктивність і можливість повторного використання коду.

Таким чином, ESLToggleable є логічним продовженням архітектури, закладеної в ESLBaseElement, і демонструють переваги використання кастомних елементів у сучасних веб-додатках.

2. Наступним елементом було створено ESLPanel (Додаток Д), який служить обгорткою для контенту, що може бути показаний або прихований. Цей компонент забезпечує можливість використання анімацій розгортання/згортання, що забезпечує плавну зміну висоти елемента, створюючи приємний користувацький досвід.

ESLPanel - це елемент, що дозволяє вбудованому контенту з’являтися або ховатися за потреби. Він може використовуватися для створення динамічних інтерфейсів, де необхідно керувати видимістю контенту без перезавантаження сторінки. Завдяки плавним анімаціям зміни висоти, користувачі отримують відчуття природної взаємодії з інтерфейсом, що покращує загальне сприйняття застосунку.

Одна з головних особливостей ESLPanel - це підтримка анімацій при розгортанні або згортанні контенту. Використовуючи плавні переходи висоти, елемент забезпечує естетично приємний спосіб показу або приховування інформації, що робить його зручним для використання в

інтерфейсах, де є багато додаткової або прихованої інформації (наприклад, аккордеони, списки FAQ тощо) (рис. 3.4).

Рисунок 3.4: Елемент ESLPanel

3. Ще одним елементом, створеним на базі ESLToggleable, є ESLSelectList (Додаток Е). Цей компонент використовується для відображення списку елементів, які можна вибирати, прикрашаючи вбудований елемент HTMLSelectElemet.

ESLSelectList - це компонент, що дозволяє користувачу вибирати елементи зі списку. Він надає покращений інтерфейс взаємодії з можливістю налаштовування зовнішнього вигляду та поведінки. Однією з основних переваг ESLSelectList є його сумісність із формами HTML5, оскільки він використовує HTMLSelectElement як модель даних. Це забезпечує просту інтеграцію компонента в існуючі форми та системи збирання даних (рис. 3.5).

Рисунок 3.5: Елемент ESLSelectList

4. ESLAnimate (Додаток Є) - це модуль, який надає сервіс та DOM API для анімації елементів при їхньому перетині з областю видимості. Щоб анімація працювала, елемент повинен бути спостережуваним через ESLAnimateService, який є основою цього модуля. Такий підхід забезпечує високу продуктивність, оскільки анімації виконуються тільки тоді, коли елемент знаходиться у видимій зоні екрану. Це дозволяє оптимізувати ресурси браузера та покращує взаємодію користувача з контентом (рис. 3.6).

Рисунок 3.6: Елемент ESLAnimate

5. ESLCarousel (Додаток Ж) - це компонент слайдшоу, який дозволяє переміщатися між слайдами. Цей елемент надає зручний спосіб відображення слайдів або картинок у циклічному режимі, що може бути корисним для створення галерей, презентацій або рекламних блоків на веб-сторінках. ESLCarousel забезпечує плавне перемикання між слайдами та підтримує різні анімаційні ефекти для покращення візуального сприйняття (рис. 3.7).

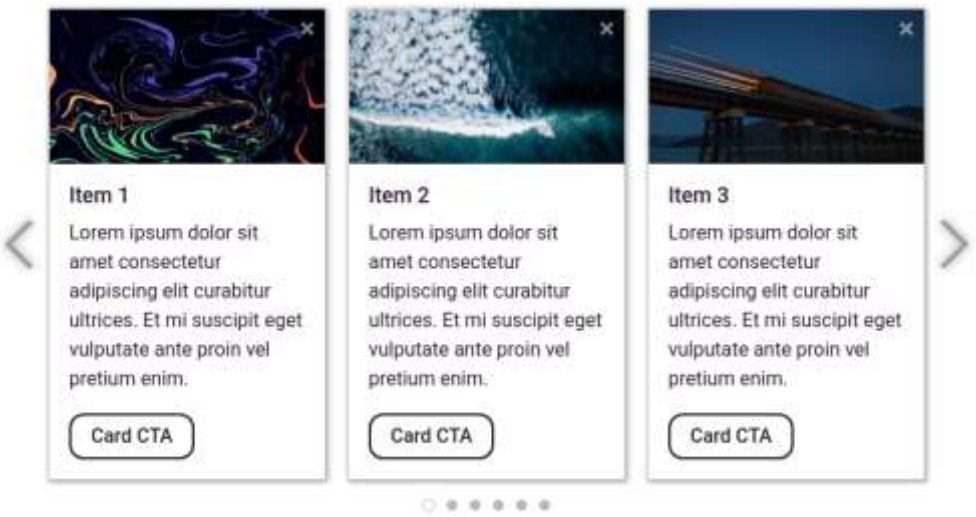


Рисунок 3.7: Елемент ESLCarousel

Найважливіші елементи бібліотеки, такі як ESLToggleable, ESLPanel, ESLSelectList, ESLAnimate та ESLCarousel, демонструють основні можливості та функціонал. Загалом у бібліотеці ESL нараховується 24 різноманітні елементи, кожен з яких спрямований на вирішення конкретних завдань у процесі розробки користувацьких інтерфейсів.

3.3. Тестування компонентів бібліотеки

Тестування та оптимізація є ключовими етапами у процесі розробки будь-якої бібліотеки користувацьких інтерфейсів, і бібліотека ESL не є

винятком. Ці етапи дозволяють забезпечити стабільну роботу компонентів у різних середовищах, їхню сумісність із різними браузерами та платформами, а також гарантують, що бібліотека залишається легкою та швидкодіючою навіть при додаванні складного функціоналу.

Тестування бібліотеки компонентів включає кілька важливих аспектів, серед яких функціональне тестування, тестування продуктивності та тестування на сумісність:

- Основна мета функціонального тестування - переконатися, що кожен компонент працює так, як очікується, та відповідає вимогам специфікацій. Для цього використовуються автоматизовані тести, які перевіряють різні сценарії використання компонентів. У бібліотеці ESL для цього застосовуються такі інструменти, як Jest та Puppeteer:

- Jest виконує модульні тести, що перевіряють логіку кожного компонента окремо, а також його взаємодію з іншими компонентами;

- Puppeteer використовується для e2e тестування, яке перевіряє роботу компонентів у браузерному середовищі, симулюючи взаємодію з користувачем шляхом генерації зображення перед змінами в коді та після та їх порівняння (рис. 3.8.).

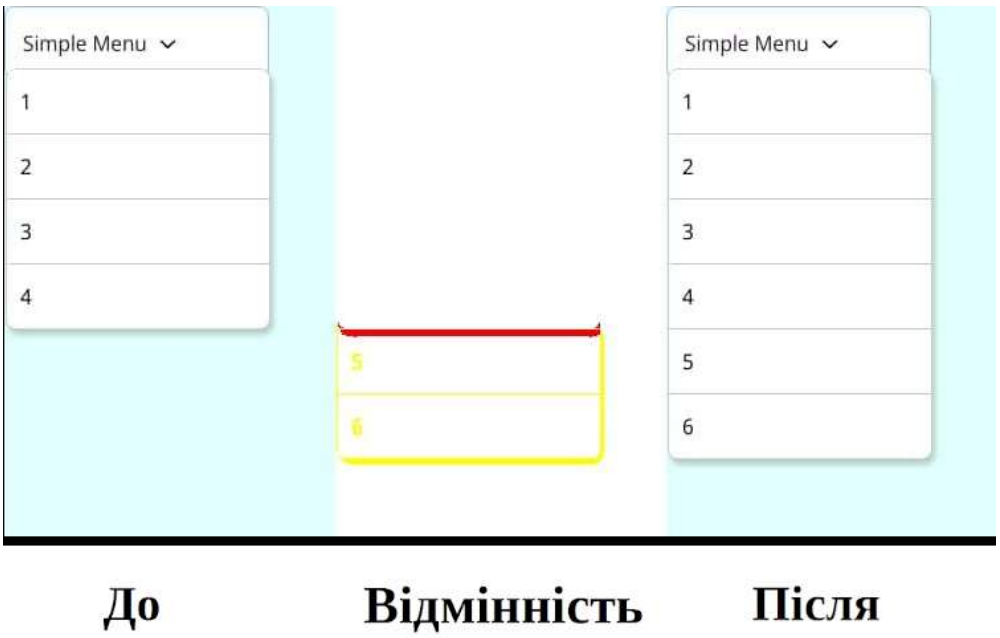


Рисунок 3.8: e2e тестування. Відмінності до та після змін

- Оскільки користувачі можуть використовувати різні браузери та пристрої, важливо перевіряти роботу компонентів у різних середовищах. Компоненти ESL проходять тестування в таких браузерах, як Google Chrome, Firefox, Safari, та Microsoft Edge. Це дозволяє забезпечити їхню коректну роботу незалежно від обраного користувачем браузера.

Для забезпечення сумісності компонентів з різними браузерами та пристроями було використано сервіс BrowserStack. За його допомогою ESL була протестована в реальних середовищах, що дало можливість перевірити роботу компонентів на численних операційних системах і версіях браузерів. Це забезпечило впевненість у коректній роботі компонентів на всіх популярних платформах (рис. 3.9).

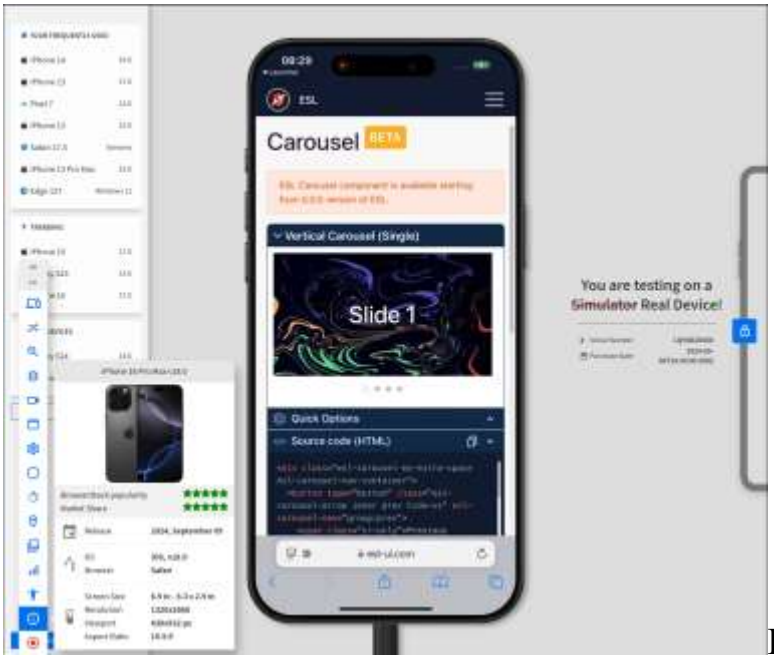
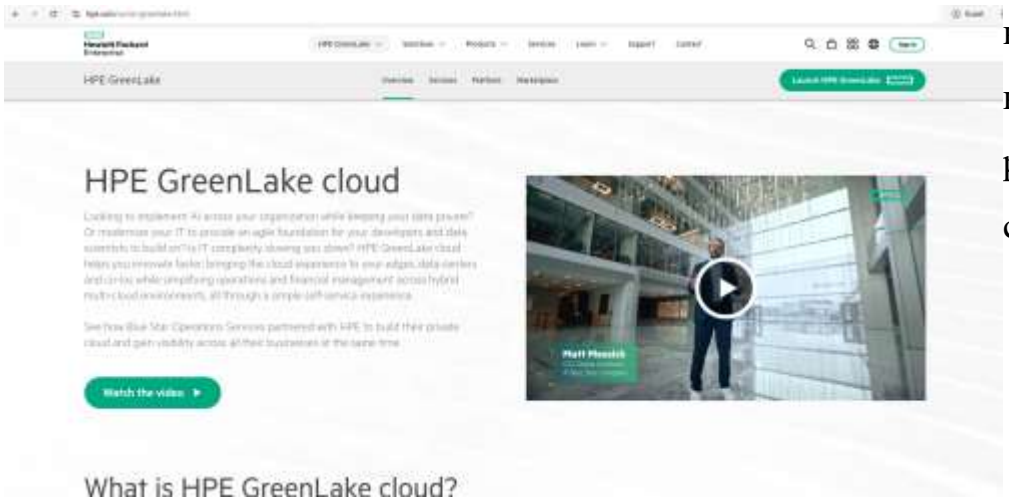


Рисунок 3.9: Тестування за допомогою BrowserStack

- У процесі тестування бібліотеки компонентів важливим етапом стало тестування на реальному проєкті. Для цього були залучені елементи ESL у рамках вебсайту “Hewlett Packard Enterprise” (рис. 3.10), що дозволило

провести всебічну перевірку компонентів у реальних умовах. Це тестування включало не лише функціональну перевірку, але й оцінку продуктивності та стабільності під навантаженням у робочому середовищі.

Рисунок 3.10 Hewlett Packard



Основною метою цього етапу було переконатися, що компоненти ESL працюють коректно в реальних умовах використання, відповідають вимогам до продуктивності та не спричиняють додаткових затримок під час завантаження сторінок. Вебсайт hpe.com, який обслуговує мільйони користувачів, став ідеальною платформою для такого тестування, оскільки дозволив оцінити поведінку компонентів ESL під реальним навантаженням:

- Тестування включало аналіз швидкості завантаження та відгуку компонентів під час взаємодії користувачів із сайтом;

- Перевірялася сумісність з іншими елементами сайту, а також відсутність конфліктів зі стилями та скриптами сторонніх бібліотек;

- Були задіяні інструменти моніторингу продуктивності для оцінки впливу компонентів на загальний час завантаження сторінки та відгук інтерфейсу при взаємодії з користувачами.

Тестування на сайті hpe.com проводилося за допомогою сервісу Google PageSpeed Insights і охоплювало кілька важливих аспектів, що стосуються

аналізу швидкості завантаження та реакції компонентів під час взаємодії користувачів. Однією з основних цілей було визначити, як швидко користувачі можуть отримати доступ до основних функцій сайту і як оперативно реагують різні елементи інтерфейсу.

У процесі аналізу виявлено, що основною причиною візуального блокування стало завантаження зображень з надмірно великим розширенням, що негативно вплинуло на швидкість відображення візуальних елементів. При цьому бібліотека ESL, яка використовується на сайті, не стала джерелом цієї проблеми (рис. 3.11).

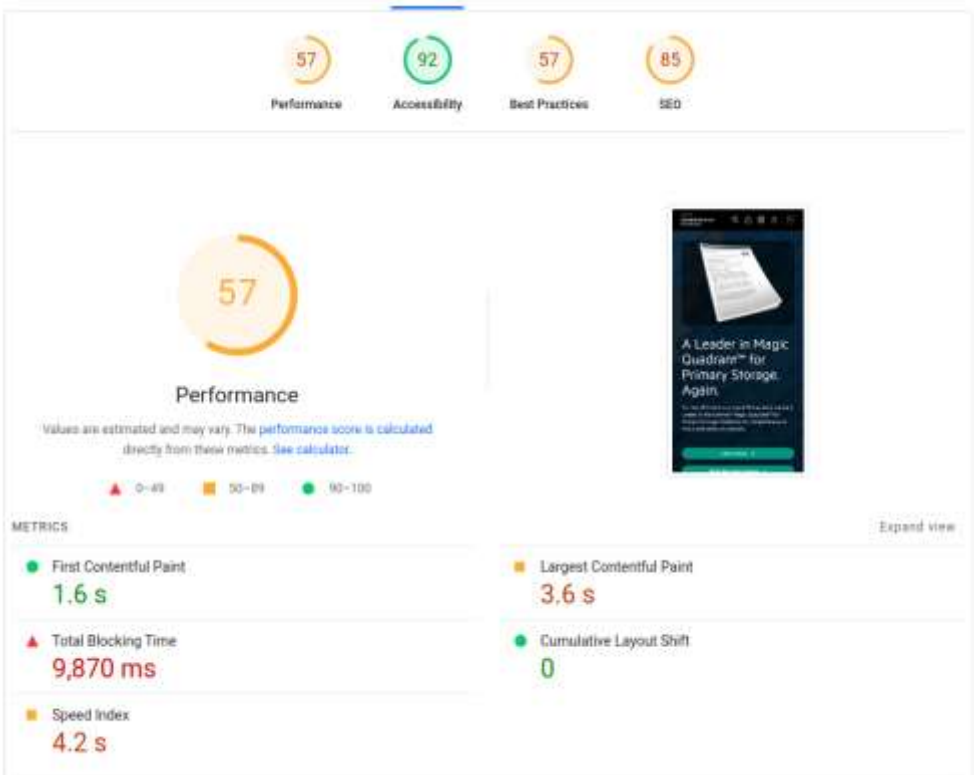


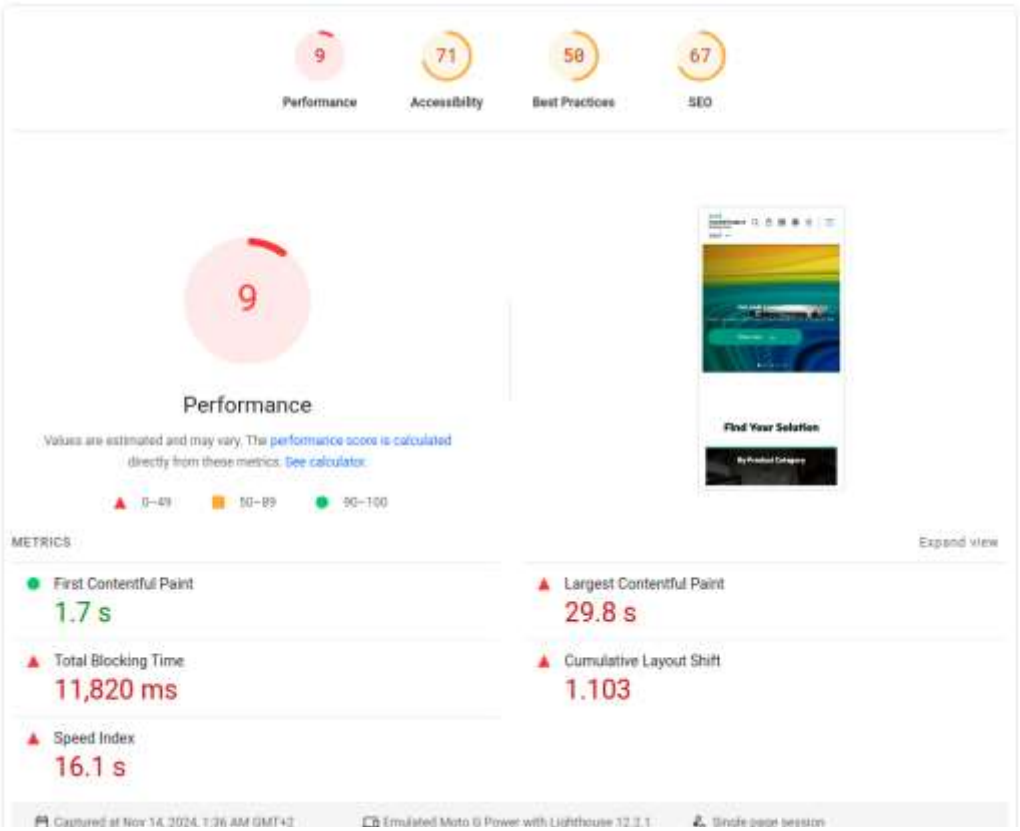
Рисунок 3.11 Аналіз продуктивності сайту hpe.com

Результати показали, що сайт hpe.com є досить важким і в середньому використовує близько 10 секунд для завантаження всіх скриптів. Однак варто

зазначити, що завантаження бібліотеки ESL зайняло лише 0.6 секунди, що свідчить про її ефективність. Найбільшою проблемою на сайті виявилося повільне завантаження скриптів, а аналіз за допомогою PageSpeed Insights виявив найбільш важкі та повільні скрипти. Водночас бібліотека ESL не була серед цих важких компонентів, що підкреслює її оптимізацію та ефективність у загальному контексті продуктивності сайту.

Для наочнішого порівняння було проаналізовано сайт hpe.com та аналогічний за навантаженням і кількістю скриптів сайт buy.hpe.com, платформу для покупки обладнання та послуг “Hewlett Packard Enterprise”. На відміну від основного сайту, замість бібліотеки ESL тут використовується бібліотека Bootstrap. Якщо провести тестування платформи за допомогою PageSpeed Insights, то виявляється що на сайті набагато гірші показники (рис. 3.12).

Рисунок 3.12 Аналіз продуктивності сайту buy.hpe.com



Сервіс PageSpeed Insights надає багатогранний аналіз швидкості сторінок, який наведений у (табл. 3.2).

Таблиця 3.2

Порівняння швидкості роботи сайтів

Бібліотека	hpe.com (ESL)	buy.hpe.com (Bootstrap)
Продуктивність	57%	9%
Доступність	92%	71%
Найкращі практики	57%	50%

Продовження таблиці 3.2

SEO	85%	67%
Перше відображення контенту	1.6 с	1.7 с
Найбільше відображення контенту	3.6 с	29.8 с
Загальний час блокування	9,870 мс	11,820 мс
Індекс швидкості	4.2 с	16.1 с
Кумулятивне зміщення макету	0 с	1.103 с

Сайт buy.hpe.com використовує Bootstrap і має гірші показники продуктивності, зокрема низьку продуктивність у 9%, у порівнянні з hpe.com, який використовує ESL і має продуктивність 57%, що є найважливішим показником в порівнянні.

На buy.hpe.com найбільші проблеми пов’язані з високим показником Найбільшим відображенням контенту (29.8 с) та Загальним часом блокування (11,820 мс), що свідчить про значне навантаження на відображення контенту.

Кумулятивне зміщення макету на buy.hpe.com також суттєво вище (1.103с) порівняно з hpe.com (0с), що може вказувати на проблеми зі стабільністю візуального контенту при завантаженні.

Тобто використання ESL на hpe.com позитивно впливає на показники, забезпечуючи швидше завантаження, менший блокувальний час та стабільніший інтерфейс.

3.4. Оптимізація компонентів

Оптимізація компонентів - це процес, спрямований на зменшення навантаження на браузер та прискорення роботи компонентів. Основні стратегії оптимізації включають мінімізацію розміру коду, зменшення кількості запитів до сервера та використання сучасних технологій для підвищення продуктивності.

Мінімізація коду. Використання таких інструментів, як Webpack дозволило мінімізувати та об'єднати код компонентів, зменшуючи розмір бібліотеки і тим самим прискорюючи завантаження сторінки. Крім того, TypeScript використовується для поліпшення структурованості коду і забезпечення кращої продуктивності під час його виконання.

Для управління стилями бібліотека ESL використовує препроцесор Less, який дозволяє організувати та мінімізувати CSS-код. Використання препроцесора сприяє швидшій обробці стилів, а також зменшує їхній розмір завдяки автоматичній генерації компактного CSS-коду.

Щоб уникнути зайвого навантаження на систему, бібліотека ESL реалізує підхід "ледачого завантаження", коли компоненти завантажуються тільки тоді, коли вони дійсно потрібні на сторінці. Це дозволяє значно зменшити час завантаження веб-сторінок і підвищити швидкодію.

Анімаційні оптимізації. Компоненти з анімаційними ефектами, як-от ESLAnimate, оптимізуються для забезпечення плавного відтворення анімацій. Для цього використовуються властивості CSS і Intersection Observer API, що дозволяють виконувати анімації тільки тоді, коли елемент з’являється у видимій частині екрана.

Процес тестування та оптимізації компонентів бібліотеки ESL дозволяє досягти високої надійності, продуктивності та сумісності. Використання сучасних інструментів для автоматизації тестування гарантує, що кожен компонент працює належним чином у різних середовищах і за різних умов. Завдяки оптимізації продуктивності бібліотека залишається легкою, швидкодіючою і відповідає вимогам сучасних веб-додатків.

3.5. Порівняння ефективності власної бібліотеки з існуючими рішеннями

Порівняння ефективності власної UI бібліотеки з існуючими рішеннями є важливим етапом у визначенні її конкурентних переваг і недоліків. Для оцінки ефективності бібліотеки використовуються наведені нижче метрики, які дозволяють об'єктивно порівняти ключові характеристики, що впливають на продуктивність, зручність використання, а також вплив на кінцевий користувацький досвід з метою порівняння власної бібліотеки з такими популярними бібліотеками, як Material-UI, Bootstrap, та Tailwind CSS.

Для більш об'єктивного порівняння власної бібліотеки з популярними UI рішеннями були визначені кілька ключових метрик, які допоможуть проаналізувати переваги та недоліки кожного з рішень. Ці метрики включають технічні аспекти, такі як розмір пакету, продуктивність та кастомізація, а також більш практичні показники, як ціна та універсальність. Нижче детально розглянуті кожна з цих метрик:

1. Розмір пакету визначає кількість даних, яку необхідно завантажити, щоб підключити бібліотеку до проекту. Чим менший розмір пакету, тим швидше сторінка завантажується. Малі бібліотеки споживають менше ресурсів і забезпечують кращий користувацький досвід, особливо на повільних мережах або мобільних пристроях. Великі бібліотеки можуть бути важчими для завантаження і можуть уповільнити роботу веб-додатку, особливо якщо використовуються лише окремі компоненти з бібліотеки.

2. Продуктивність відображає швидкість рендерингу компонентів, обробку їх взаємодії з користувачем та загальний вплив на ефективність роботи веб-застосунку. Високопродуктивні бібліотеки швидко малюють компоненти та зменшують затримки у користувацьких взаємодіях, що робить інтерфейс плавним і чуйним. Більш важкі бібліотеки або бібліотеки з великою кількістю стилів та логіки можуть знижувати продуктивність за рахунок складності та кількості даних, які треба обробити.

3. Гнучкість та кастомізація визначає, наскільки легко розробникам налаштувати компоненти під свої специфічні вимоги. Висока гнучкість дозволяє кастомізувати компоненти з мінімальними зусиллями, що є особливо важливим для створення унікальних інтерфейсів. Деякі бібліотеки надають широкі можливості кастомізації через змінні стилів, тоді як інші можуть бути більш жорстко обмежені фіксованим дизайном або структурою компонентів.

4. Сумісність із браузерами є важливим показником для UI бібліотек, оскільки користувачі можуть використовувати різні браузери для доступу до веб-застосунків. Важливо, щоб компоненти бібліотеки однаково добре працювали в усіх основних браузерах, таких як Google Chrome, Mozilla Firefox, Safari, і Microsoft Edge. Бібліотеки, що не забезпечують крос-браузерну підтримку, можуть викликати проблеми у користувачів на різних платформах.

5. Легкість використання оцінює, наскільки просто розробникам інтегрувати бібліотеку в проект і почати з нею працювати. Бібліотеки, які мають добре продуману документацію, простий API і інтуїтивно зрозумілий синтаксис, зазвичай легше у використанні, що зменшує криву навчання для новачків. Складніші бібліотеки можуть мати більш круту криву навчання, але при цьому надавати більші можливості.

6. Підтримка мобільних пристроїв та адаптивність визначає здатність інтерфейсу змінювати свій вигляд та структуру відповідно до розміру екрана або типу пристрою. Це особливо важливо для мобільних додатків і веб-сайтів, які повинні виглядати добре як на великих екранах, так і на мобільних телефонах. Бібліотеки, що мають вбудовану підтримку адаптивності, дозволяють легко створювати мобільні та планшетні версії інтерфейсів.

7. Кількість елементів відображає кількість компонентів, доступних у бібліотеці. Чим більше кастомних елементів бібліотека пропонує, тим більше варіантів готових рішень для різних завдань. Наявність великого набору компонентів дозволяє швидко інтегрувати готові рішення, не витрачаючи час на їх розробку з нуля. Деякі UI бібліотеки не пропонують жодного елемента, а навпка акцентують увагу на стилістичних рішеннях для вбудованих HTML елементів.

8. Ціна є важливою метрикою, оскільки багато бібліотек мають безкоштовні або умовно безкоштовні версії, але можуть вимагати платної

ліцензії для комерційного або корпоративного використання. Деякі бібліотеки доступні лише у платних версіях для великих підприємств, що може бути значущим фактором при виборі рішення. Безкоштовна ліцензія є привабливою для малих та середніх проектів. Платна ліцензія може бути необхідною для корпоративного використання або для отримання розширеної підтримки.

9. Універсальність вказує на можливість використання бібліотеки з різними фреймворками або платформами. Деякі бібліотеки, як-от Material-UI, розраховані на роботу тільки з конкретними фреймворками (наприклад, React), тоді як інші є універсальними та можуть працювати на будь-яких платформах. Чим універсальніша бібліотека, тим легше її інтегрувати в різноманітні проекти з різними технологічними стеками.

У рамках порівняння були використані раніше згадані популярні UI бібліотеки, і їхні характеристики були співставлені з характеристиками ESL для оцінки продуктивності та функціональності (табл. 3.3).

Таблиця 3.3

Порівняльна характеристика бібліотек

Метрика	Характеристика	Власна бібліотека ESL	Material-UI	Bootstrap	Tailwind CSS	JQuery-UI	Semantic-UI	Ant Design	Chakra-UI	Vuetify
Розмір пакету	Розмір пакету	Малий (1.13 MB)	Великий (7.02 MB)	Великий (9.67 MB)	Середній (5.71 MB)	Середній (4.56 MB)	Великий (13.2 MB)	Дуже великий (46.7 MB)	Малий (0.9 MB)	Дуже великий (36 MB)
Продуктивність	Оцінка продуктивності (1-5)	5	3	3	5	2	2	2	4	3
Гнучкість та кастомізація	Оцінка можливостей кастомізації та адаптації (1-5)	5	4	2	5	4	4	3	4	1
Сумісність із різними браузерами	Оцінка сумісності (1-5)	5	4	4	4	3	4	4	4	4

Легкість використання	Оцінка зручності інтеграції (1-5)	5	3	5	2	5	3	3	5	4
Підтримка мобільних пристроїв	Оцінка адаптивності (1-5)	5	5	5	4	1	4	5	5	5

Продовження таблиці 3.3

Кількість кастомних елементів	Оцінка функціональності (кількість елементів)	24	50	24	Немає власних елементів	20	46	69	22	66
Ціна	Вартість використання	Безкоштовна	Безкоштовна (платна для корпорацій)	Безкоштовна	Безкоштовна	Безкоштовна	Безкоштовна	Безкоштовна (платна для корпорацій)	Безкоштовна	Безкоштовна (платна для корпорацій)
Універсальність	Оцінка можливості використання в різних фреймворках (1 - обмежена, 5 - універсальна)	5	1	5	5	5	5	3	1	2

Виходячи з характаристики, можна оцінити ефективність кожної бібліотеки у відсотках, враховуючи середню оцінку за всіма метриками для кожної з них за формулою 3.1:

$$V = \left(\frac{\sum_i C_i}{\sum_i M_i} \right) \times 100\% \quad (3.1)$$

де:

V - Відсоткова оцінка продуктивності бібліотеки;

□□ - максимальна можлива сума оцінок метрик;

С і - сума оцінок метрик для бібліотеки.

Розрахунки виконані за формулою відсоткової оцінки для визначення продуктивності кожної бібліотеки (табл. 3.4).

Таблиця 3.4

Порівняльна таблиця відсоткових оцінок бібліотек

Бібліотека	Відсоткова оцінка
ESL	91.25%
Material-UI	77.5%
Bootstrap	75%
Tailwind CSS	76.25%
JQuery-UI	61.25%
Semantic-UI	76.25%
Ant Design	72.5%
Chakra-UI	83.75%
Vuetify	65%

З таблиці видно, що власна бібліотека має оцінкову перевагу у 91%. На практиці бібліотека має переваги в аспектах продуктивності, легкості використання та універсальності. Вона є безкоштовною і підтримує роботу на різних платформах, на відміну від деяких бібліотек, таких як Material-UI та Ant Design, які орієнтовані на React. Також власна бібліотека виділяється меншим розміром пакету, що робить її оптимальним вибором як малих та і великих проектів. Бібліотеки, як Semantic-UI та Vuetify, забезпечують великий набір компонентів, але можуть бути менш продуктивними через більший розмір та складність.

Варто відзначити, що хоча Chakra-UI також має високу гнучкість, вона орієнтована лише на React. В цілому, ESL пропонує більш універсальний

підхід і сумісність з різними платформами, що забезпечує її високий відсотковий рейтинг у порівнянні з іншими популярними бібліотеками.

Висновки за розділом 3

Розробка проекту ESL була зумовлена необхідністю створення легкої та продуктивної альтернативи наявним рішенням, які зазвичай зосереджені на стилізації та складних анімаціях. Бібліотека була спроектована з акцентом на необхідну функціональність для корпоративних вебсайтів, мінімізуючи навантаження на ресурси системи і уникаючи важких фреймворків, таких як React.

У процесі розробки було обрано ряд технологій, включно з TypeScript для підвищення надійності коду, Less для управління стилями, а також інструменти для контролю версій та забезпечення якості коду, такі як GIT, NPM, Webpack, ESLint, Jest і Puppeteer. Архітектурною основою бібліотеки став клас ESLBaseElement, що використовує Web API для створення користувацьких веб-компонентів, таких як ESLToggleable, ESLPanel і ESLCarousel, які відповідають вимогам інтерактивності корпоративних сайтів.

Порівняння з популярними бібліотеками підтвердило переваги власної бібліотеки в аспектах продуктивності, простоти використання та менших розмірах пакету. Вона забезпечує гнучкість і кастомізацію, зокрема через компоненти, які мають складнішу функціональність. Загалом, розробка цієї бібліотеки вирішує критичні питання, пов’язані з продуктивністю та адаптивним дизайном, роблячи її ідеальним інструментом для сучасних корпоративних вебсайтів.

ВИСНОВОК

Використання UI-бібліотек є важливим аспектом розробки сучасних вебзастосунків і мобільних додатків, оскільки вони забезпечують стандартизовані рішення для створення інтерфейсів користувача. Це не лише полегшує роботу розробників, а й прискорює процес створення продуктів, дозволяючи зосередитися на бізнес-логіці та функціональності. UI-бібліотеки впливають на всі етапи розробки, від прототипування до інтеграції в готові системи, сприяючи стандартизації інтерфейсів і покращенню користувацького досвіду (UX). При цьому правильний вибір бібліотеки залежить від типу проєкту, технологічного стека, необхідної продуктивності та гнучкості. У ході дослідження:

- Розглянуто теоретичні аспекти застосування UI-бібліотек, що дозволило визначити їх переваги, недоліки та вплив на процес розробки. Зокрема, UI-бібліотеки прискорюють створення продукту завдяки готовим компонентам і шаблонам, стандартизують інтерфейси, покращують доступність та забезпечують зручність для користувачів. Водночас важливо враховувати обмеження, такі як знижена гнучкість дизайну та можливі труднощі з інтеграцією в специфічні проєкти. Огляд популярних бібліотек, таких як React, Bootstrap та Material-UI, продемонстрував їхній широкий функціонал, але також виявив обмеження, такі як складність налаштувань та можливе перевантаження додатків зайвими компонентами.

- Вивчено вплив UI-бібліотек на розробку програмного забезпечення на прикладі створення додатку з використанням як UI-бібліотеки, так і стандартних елементів HTML. Це порівняння дозволило оцінити, як різні підходи впливають на швидкість розробки, підтримку, продуктивність і гнучкість проєкту. У контексті вибору UI-бібліотеки було вивчено ключові фактори, такі як сумісність із технологічним стеком, доступність

документації, продуктивність та можливості кастомізації. Легкі бібліотеки підходять для простих проєктів, де критично важливими є швидкість завантаження та мінімізація використаних ресурсів. У свою чергу, для складніших проєктів, наприклад корпоративних рішень, необхідні потужніші й гнучкіші бібліотеки, що забезпечують можливість реалізації більш складних функцій та інтеграцій.

- Розроблено власну UI-бібліотеку ESL, створену для вирішення корпоративних завдань. Вона забезпечує високу продуктивність, гнучкість, мінімізацію ресурсних витрат та адаптивність до проєктних вимог. У бібліотеку інтегровано 24 компоненти, що охоплюють як базові, так і складні функції, наприклад, ESLAnimate для анімацій або ESLCarousel для слайд-шоу.

- Проведено оцінку ефективності власної бібліотеки ESL у порівнянні з популярними бібліотеками, такими як Material-UI, Bootstrap, Tailwind CSS, Chakra-UI тощо. Результати показали, що ESL отримала найвищу відсоткову оцінку - 91.25%, тоді як найближчі конкуренти Chakra-UI та Material-UI досягли 83.75% і 77.5% відповідно. Особливі переваги ESL полягають у продуктивності, універсальності та малому розмірі пакету (1.13 MB), що перевищує показники конкурентів, таких як Material-UI (7.02 MB) і Tailwind CSS (5.71 MB).

- Тестування на реальному проєкті підтвердило високу ефективність бібліотеки ESL. На сайті hpe.com, який використовує ESL, продуктивність досягла 57%, що значно перевищує показник 9% на сайті buy.hpe.com, де застосовується бібліотека Bootstrap. Ці результати демонструють переваги ESL у забезпеченні продуктивності, стабільності та швидкості завантаження контенту, що є критично важливим для корпоративних вебзастосунків.

UI-бібліотеки є не лише інструментом, а стратегічною складовою сучасного процесу розробки. Розроблена бібліотека ESL довела свою

ефективність у реальних умовах, забезпечуючи гнучкість, продуктивність та зручність використання. Її переваги роблять її перспективним рішенням для подальшого використання та вдосконалення.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gothelf J. Lean UX: Applying Lean Principles to Improve User Experience. Німеччина: O'Reilly Media, 2013. 44 с.

2. Shenoy A., Sossou U. Learning Bootstrap. Велика Британія: Packt Publishing, 2015.

3. Marcel Souza. Mastering javascript complete course. Бразилія: Gavea, 2024. 106 с.

4. Brown T. Comparing Modern CSS Solutions (Tailwind vs MUI vs Bootstrap vs Chakra). 2022. URL: https://www.youtube.com/watch?v=CQuTF-bkOgc (дата звернення: 28.09.2024).

5. Куттіг О.Б. Професійний React Native: експертні методи та рішення для створення високоякісних кросплатформних готових додатків. Велика Британія: Packt Publishing, 2022. 165 с.

6. Суф’ян У., Клауд Н., Амблер Т. Фреймворки JavaScript для сучасної веб-розробки: основні фреймворки, бібліотеки та інструменти для правильного навчання. Австралія: Apress, 2015. 20 с.

7. Хейлманн К. Початок JavaScript зі сценаріями DOM і Ajax: від новачка до професіонала. Німеччина: Apress, 2006.

8. Сейболд К. jQuery Mobile: розробка програм для смартфонів та планшетів. Німеччина: O'Reilly Media, 2013.

9. Morten B. React in Depth. США: Manning, 2024, 7 с.

10. Norman D. The Design of Everyday Things. США: Basic Books, 2002.

11. Nielsen J. Usability Engineering. США: Morgan Kaufmann, 1993.

12. Raskin D. The Humane Interface. США: Addison-Wesley Professional , 2000.

13. Hindman M. The Internet Trap: How the Digital Economy Builds Monopolies and Undermines Democracy. США: Princeton University Press, 2020, 27 с.

14. Kahney L., Ive J. The Genius Behind Apple’s Greatest Products. США: Penguin Books Limited, 2013, 32 с.

15. Anderson J. Effective UI: The Art of Building Great User Experience in Software. США: O'Reilly Media, 2010, 250 с.

16. Мартін Р. С. Чистий код. Створення і рефакторинг за допомогою Agile. США: O'Reilly Media, 2008, 346 с.

17. Фрімен Е., Робсон Е., Head First. Патерни проєктування. США: O'Reilly Media, 2004, 221 с.

18. Anderson R. Security Engineering: A Guide to Building Dependable Distributed Systems. США: Wiley, 2008, 157 с.

19. Рохас С. Створення прогресивних веб-додатків за допомогою Vue.js: надійні, швидкі та цікаві програми за допомогою Vue.js. Німеччина: Apress, 2019, 60 с.

20. Chacon S., Straub B. Pro Git. США: Apress, 2014, 5 с.

21. Mastering TypeScript. N.p., Cybellium Ltd, 2023.

22. Бузід М. Webpack для початківців: ваш покроковий посібник із вивчення Webpack. Німеччина: Apress, 2020.

23. Шовчко Ф. Д. Impact Analysis of UI Libraries on Developer and User Experience. Київ: Пленарне засідання ЕПФ, 2024.

ДОДАТКИ

Додаток А

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Sign in</title>
   <link rel="stylesheet" href="styles.css">
</head>
<body>
   <div class="container">
      <div class="form-box">
         <h2>Sign in</h2>
         <p>Stay updated on your professional world</p>
         <form id="loginForm">
            <label for="email">Email or Phone</label>
            <input type="text" id="email" name="email" placeholder="Email or Phone"
required>

            <label for="password">Password</label>
            <div class="password-wrapper">
               <input type="password" id="password" name="password"
placeholder="Password" required>
               <span id="show-password" onclick="togglePassword()">show</span>
            </div>

            <a href="#" class="forgot-link">Forgot password?</a>

            <button type="submit" class="sign-in-btn">Sign in</button>

            <div class="divider">or</div>

            <button type="button" class="google-sign-in-btn">
               <img src="google-logo.png" alt="Google logo"> Sign in with Google
```

```
            </button>
          </form>
        </div>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

index.css

```
* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.container {
    display: flex;
    justify-content: center;
    align-items: center;
    width: 100%;
}

.form-box {
    background-color: white;
    padding: 40px;
    border-radius: 8px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
    max-width: 400px;
```

```
    width: 100%;
    text-align: center;
}

h2 {
    font-size: 24px;
    margin-bottom: 10px;
}

p {
    font-size: 14px;
    color: #666;
    margin-bottom: 20px;
}

label {
    display: block;
    text-align: left;
    font-size: 14px;
    margin-bottom: 5px;
}

input {
    width: 100%;
    padding: 10px;
    margin-bottom: 20px;
    border: 1px solid #ccc;
    border-radius: 4px;
}

.password-wrapper {
    position: relative;
}

.password-wrapper #show-password {
    position: absolute;
    right: 10px;
    top: 50%;
```

```
    transform: translateY(-50%);
    cursor: pointer;
    color: #0073b1;
}

.forgot-link {
    display: block;
    text-align: right;
    font-size: 12px;
    color: #0073b1;
    text-decoration: none;
    margin-bottom: 20px;
}

.forgot-link:hover {
    text-decoration: underline;
}

.sign-in-btn {
    width: 100%;
    background-color: #0073b1;
    color: white;
    padding: 10px;
    border: none;
    border-radius: 4px;
    font-size: 16px;
    cursor: pointer;
    margin-bottom: 10px;
}

.sign-in-btn:hover {
    background-color: #005580;
}

.divider {
    margin: 20px 0;
    font-size: 14px;
    color: #666;
```

```
    position: relative;
}

.divider::before,
.divider::after {
    content: "";
    position: absolute;
    top: 50%;
    width: 40%;
    height: 1px;
    background-color: #ccc;
}

.divider::before {
    left: 0;
}

.divider::after {
    right: 0;
}

.google-sign-in-btn {
    width: 100%;
    background-color: black;
    color: white;
    padding: 10px;
    border: none;
    border-radius: 4px;
    font-size: 16px;
    cursor: pointer;
    display: flex;
    align-items: center;
    justify-content: center;
}

.google -sign-in-btn img {
    width: 20px;
    margin-right: 10px;
```

```
}

.google -sign-in-btn:hover {
   background-color: #333;
}
```

script.js

```
function togglePassword() {
   const passwordField = document.getElementById('password');
   const showPasswordText = document.getElementById('show-password');

   if (passwordField.type === 'password') {
      passwordField.type = 'text';
      showPasswordText.textContent = 'hide';
   } else {
      passwordField.type = 'password';
      showPasswordText.textContent = 'show';
   }
}

function validateEmailOrPhone(input) {
   const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
   const phonePattern = /^\+?[0-9]{7,15}$/;

   return emailPattern.test(input) || phonePattern.test(input);
}

function showError(message) {
   const errorBox = document.getElementById('error-message');
   if (errorBox) {
      errorBox.textContent = message;
      errorBox.style.display = 'block';
   }
}

function hideError() {
   const errorBox = document.getElementById('error-message');
   if (errorBox) {
```

```
        errorBox.style.display = 'none';
    }
}

function validateForm(event) {
    event.preventDefault();

    const emailInput = document.getElementById('email').value.trim();
    const passwordInput = document.getElementById('password').value.trim();

    if (emailInput === '' || passwordInput === '') {
        showError('Both fields are required.');
        return;
    }

    if (!validateEmailOrPhone(emailInput)) {
        showError('Please enter a valid email or phone number.');
        return;
    }
    hideError();

    console.log('Form submitted successfully!');
    document.getElementById('loginForm').submit();
}

document.getElementById('loginForm').addEventListener('submit', validateForm);
```

Додаток Б

index.html

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

```
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
        <style>
            .container {
                height: 100vh;
                display: flex;
                justify-content: center;
                align-items: center;
            }

            .form-box {
                background-color: white;
                padding: 40px;
                border-radius: 8px;
                box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
                max-width: 400px;
                width: 100%;
                text-align: center;
            }

            .password-wrapper {
                position: relative;
            }

            .password-wrapper #show-password {
                position: absolute;
                right: 10px;
                top: 50%;
                transform: translateY(-50%);
                cursor: pointer;
                color: #0073b1;
            }

            .divider {
                position: relative;
                text-align: center;
                margin: 20px 0;
            }
```

```
            .divider::before,
            .divider::after {
               content: "";
               position: absolute;
               top: 50%;
               width: 40%;
               height: 1px;
               background-color: #ccc;
            }

            .divider::before {
               left: 0;
            }

            .divider::after {
               right: 0;
            }
         </style>
      </head>
      <body>
         <div class="container">
            <div class="form-box">
               <h2>Sign in</h2>
               <p>Stay updated on your professional world</p>

               <div id="error-message" class="alert alert-danger" style="display:
none;"></div>

               <form id="loginForm" class="needs-validation" novalidate>
                  <div class="mb-3">
                     <label for="email" class="form-label">Email or Phone</label>
                     <input type="text" class="form-control" id="email" name="email"
placeholder="Email or Phone" required>
                     <div class="invalid-feedback">Please enter your email or phone.</div>
                  </div>

                  <div class="mb-3">
```

```
                    <label for="password" class="form-label">Password</label>
                    <div class="password-wrapper">
                        <input type="password" class="form-control" id="password"
name="password" placeholder="Password" required>
                        <span id="show-password" onclick="togglePassword()">show</span>
                    </div>
                    <div class="invalid-feedback">Please enter your password.</div>
                </div>

                <a href="#" class="d-block text-end mb-3">Forgot password?</a>

                <button type="submit" class="btn btn-primary w-100">Sign in</button>

                <div class="divider">or</div>

                <button type="button" class="btn btn-dark w-100">
                    <img src="apple-logo.png" alt="Apple logo" style="width: 20px; margin-
right: 10px;">
                    Sign in with Apple
                </button>
            </form>
        </div>
    </div>

    <!-- Підключення Bootstrap JS та залежностей -->
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
    <script src="script.js"></script>
</body>
</html>
```

script.js

```
function togglePassword() {
    const passwordField = document.getElementById('password');
    const showPasswordText = document.getElementById('show-password');

    if (passwordField.type === 'password') {
        passwordField.type = 'text';
```

```
        showPasswordText.textContent = 'hide';
    } else {
        passwordField.type = 'password';
        showPasswordText.textContent = 'show';
    }
}

function validateEmailOrPhone(input) {
    const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    const phonePattern = /^\+?[0-9]{7,15}$/;

    return emailPattern.test(input) || phonePattern.test(input);
}

function showError(message) {
    const errorBox = document.getElementById('error-message');
    if (errorBox) {
        errorBox.textContent = message;
        errorBox.style.display = 'block';
    }
}

function hideError() {
    const errorBox = document.getElementById('error-message');
    if (errorBox) {
        errorBox.style.display = 'none';
    }
}

function validateForm(event) {
    event.preventDefault();

    const emailInput = document.getElementById('email').value.trim();
    const passwordInput = document.getElementById('password').value.trim();

    if (emailInput === '' || passwordInput === '') {
        showError('Both fields are required.');
        return;
```

```
    }

    if (!validateEmailOrPhone(emailInput)) {
        showError('Please enter a valid email or phone number.');
        return;
    }
    hideError();

    console.log('Form submitted successfully!');
    document.getElementById('loginForm').submit();
}

document.getElementById('loginForm').addEventListener('submit', validateForm);
```

Додаток В

```
import {setAttr} from '../../esl-utils/dom/attr';
import {prop} from '../../esl-utils/decorators';
import {ESLEventUtils} from '../../esl-utils/dom/events';
import {CSSClassUtils} from '../../esl-utils/dom/class';

import type {
  ESLEventListener,
  ESLListenerHandler,
  ESLListenerCriteria,
  ESLListenerDescriptor
} from '../../esl-utils/dom/events';
import type {ESLBaseComponent} from '../../esl-utils/abstract/component';

/** Finalize tag name (`is`) for passed ESLBaseElement-based class */
const finalize = (type: typeof ESLBaseElement, tagName: string): void => {
  Object.defineProperty(type, 'is', {
    get: () => tagName,
    set(value) {
      if (this === type) throw Error(`[ESL]: Cannot override ${type.name}.is property, the class
is already registered`);
      Object.defineProperty(this, 'is', {value, writable: true, configurable: true});
    }
  });
```

```
};

/**
 * Base class for ESL custom elements
 * Allows defining custom element with the optional custom tag name
 */
export abstract class ESLBaseElement extends HTMLElement implements ESLBaseComponent {
  /** Custom element tag name */
  public static is = '';

  /** Event to indicate component significant state change that may affect other components state */
  @prop('esl:refresh') public REFRESH_EVENT: string;

  protected _connected: boolean = false;

  /** @returns custom element tag name */
  public get baseTagName(): string {
    return (this.constructor as typeof ESLBaseElement).is;
  }

  protected connectedCallback(): void {
    this._connected = true;
    this.classList.add(this.baseTagName);

    // Automatic subscription happens only if the element is currently in the DOM
    if (this.isConnected) ESLEventUtils.subscribe(this);
  }
  protected disconnectedCallback(): void {
    this._connected = false;

    ESLEventUtils.unsubscribe(this);
  }

  /**
   * Callback to handle changing of element attributes.
   * Happens when attribute accessed for writing independently of the actual value change
```

```
   */
  protected attributeChangedCallback(name: string, oldValue: string | null, newValue: string | null): void {}

  /** Checks that the element's `connectedCallback` has been executed */
  public get connected(): boolean {
    return this._connected;
  }

  /** Subscribes (or resubscribes) all known descriptors that matches criteria */
  public $$on(criteria: ESLListenerCriteria): ESLEventListener[];
  /** Subscribes `handler` method marked with `@listen` decorator */
  public $$on(handler: ESLListenerHandler): ESLEventListener[];
  /** Subscribes `handler` function by the passed DOM event descriptor {@link ESLListenerDescriptor} or event name */
  public $$on<EType extends keyof ESLListenerEventMap>(
    event: EType | ESLListenerDescriptor<EType>,
    handler: ESLListenerHandler<ESLListenerEventMap[EType]>
  ): ESLEventListener[];
  public $$on(event: any, handler?: any): ESLEventListener[] {
    return ESLEventUtils.subscribe(this, event, handler);
  }

  /** Unsubscribes event listener */
  public $$off(...condition: ESLListenerCriteria[]): ESLEventListener[] {
    return ESLEventUtils.unsubscribe(this, ...condition);
  }

  /**
   * Gets or sets CSS classes for the current element.
   * @param cls - CSS classes query {@link CSSClassUtils}
   * @param value - boolean to set CSS class(es) state or undefined to skip mutation
   * @returns current classes state or passed state
   */
  public $$cls(cls: string, value?: boolean): boolean {
    if (value === undefined) return CSSClassUtils.has(this, cls);
    CSSClassUtils.toggle(this, cls, value);
    return value;
```

```
  }

  /**
   * Gets or sets an attribute for the current element.
   * If the `value` param is undefined then skips mutation.
   * @param name - attribute name
   * @param value - string attribute value, boolean attribute state or `null` to delete attribute
   * @returns the current attribute value or previous value for mutation
   */
  public $$attr(name: string, value?: null | boolean | string): string | null {
    const prevValue = this.getAttribute(name);
    if (value !== undefined) setAttr(this, name, value);
    return prevValue;
  }

  /**
   * Dispatches component custom event.
   * @param eventName - event name
   * @param eventInit - custom event init. See {@link CustomEventInit}
   */
  public $$fire(eventName: string, eventInit?: CustomEventInit): boolean {
    return ESLEventUtils.dispatch(this, eventName, eventInit);
  }

  /**
   * Register component in the {@link customElements} registry
   * @param tagName - custom tag name to register custom element
   */
  public static register(this: typeof ESLBaseElement, tagName?: string): void {
    tagName = tagName || this.is;
    if (!tagName) throw new DOMException('[ESL]: Incorrect tag name',
'NotSupportedError');
    const constructor: any = customElements.get(tagName);
    if (constructor && (constructor !== this || constructor.is !== tagName)) {
      throw new DOMException('[ESL]: Element tag already occupied or inconsistent',
'NotSupportedError');
    }
    if (constructor) return;
```

```
    finalize(this, tagName);
    customElements.define(tagName, this as any as CustomElementConstructor);
  }

  /** Shortcut for `customElements.whenDefined(currentCustomElement)` */
  public static get registered(): Promise<CustomElementConstructor> {
    return customElements.whenDefined(this.is);
  }

  /** Creates an instance of the current custom element */
  public static create<T extends typeof ESLBaseElement>(this: T): InstanceType<T>;
  /** General signature of {@link create} to allow simplified overrides of the method */
  public static create(this: typeof ESLBaseElement): ESLBaseElement;
  public static create<T extends typeof ESLBaseElement>(this: T): InstanceType<T> {
    return document.createElement(this.is) as InstanceType<T>;
  }
}
```

Додаток Г

```
import {ExportNs} from '../../esl-utils/environment/export-ns';
import {ESC, SYSTEM_KEYS} from '../../esl-utils/dom/keys';
import {CSSClassUtils} from '../../esl-utils/dom/class';
import {prop, attr, jsonAttr, listen} from '../../esl-utils/decorators';
import {defined, copyDefinedKeys} from '../../esl-utils/misc/object';
import {parseBoolean, toBooleanAttribute} from '../../esl-utils/misc/format';
import {sequentialUID} from '../../esl-utils/misc/uid';
import {hasHover} from '../../esl-utils/environment/device-detector';
import {DelayedTask} from '../../esl-utils/async/delayed-task';
import {ESLBaseElement} from '../../esl-base-element/core';
import {findParent, isMatches} from '../../esl-utils/dom/traversing';

import type {DelegatedEvent} from '../../esl-event-listener/core/types';

/** Default Toggleable action params type definition */
export interface ESLToggleableActionParams {
  /** Action to execute */
```

```
  readonly action?: 'show' | 'hide';
  /** Initiator string identifier */
  initiator?: string;
  /** Delay timeout for both show and hide actions */
  delay?: number;
  /** Show delay timeout */
  showDelay?: number;
  /** Hide delay timeout */
  hideDelay?: number;
  /** Force action independently of current state of the Toggleable */
  force?: boolean;
  /** Do not throw events on action */
  silent?: boolean;
  /** Activate hover tracking to hide Toggleable */
  trackHover?: boolean;
  /** Element activator of the action */
  activator?: HTMLElement | null;
  /** Event that initiates the action */
  event?: Event;

  /** Custom user data */
  [key: string]: any;
}

export interface ESLToggleableRequestDetails extends ESLToggleableActionParams {
  // Selector to match or exact predicate to check if the target should process request
  match?: string | ((target: Element) => boolean);
}

const activators: WeakMap<ESLToggleable, HTMLElement | undefined> = new WeakMap();

@ExportNs('Toggleable')
export class ESLToggleable extends ESLBaseElement {
  public static override is = 'esl-toggleable';
  public static observedAttributes = ['open', 'group'];

  /** Default show/hide params for all ESLToggleable instances */
  public static DEFAULT_PARAMS: ESLToggleableActionParams = {};
```

```
/** Event to dispatch when toggleable is going to be activated */
@prop('esl:before:show') public BEFORE_SHOW_EVENT: string;
/** Event to dispatch when toggleable is going to be deactivated */
@prop('esl:before:hide') public BEFORE_HIDE_EVENT: string;

/** Event to dispatch when toggleable is activated */
@prop('esl:show') public SHOW_EVENT: string;
/** Event to dispatch when toggleable is deactivated */
@prop('esl:hide') public HIDE_EVENT: string;

/** Event to dispatch when toggleable has end activation process */
@prop('esl:after:show') public AFTER_SHOW_EVENT: string;
/** Event to dispatch when toggleable has end deactivation process */
@prop('esl:after:hide') public AFTER_HIDE_EVENT: string;

/** Event to activate toggleables on event way */
@prop('esl:show:request') public SHOW_REQUEST_EVENT: string;
/** Event to deactivate toggleables on event way */
@prop('esl:hide:request') public HIDE_REQUEST_EVENT: string;

/** Event to dispatch when toggleable group has changed */
@prop('esl:change:group') public GROUP_CHANGED_EVENT: string;

/**
 * CSS class (supports {@link CSSClassUtils}) to add on the body element
 * */
@attr() public bodyClass: string;
/** CSS class (supports {@link CSSClassUtils}) to add when the Toggleable is active */
@attr({defaultValue: 'open'}) public activeClass: string;

/**
 * CSS class (supports {@link CSSClassUtils}) to add/remove on the container
 * defined by {@link containerActiveClassTarget}
 */
@attr() public containerActiveClass: string;
/**
 * Selector for the closest parent element to add/remove {@link containerActiveClass}
```

```
   * (default: `*` direct parent)
   */
  @attr({defaultValue: '*'}) public containerActiveClassTarget: string;

  /** Toggleable group meta information to organize groups */
  @attr({name: 'group'}) public groupName: string;
  /** Selector to mark inner close triggers */
  @attr({name: 'close-on'}) public closeTrigger: string;

  /** Disallow automatic id creation when it's empty */
  @attr({parser: parseBoolean, serializer: toBooleanAttribute}) public noAutoId: boolean;
  /** Close the Toggleable on ESC keyboard event */
  @attr({parser: parseBoolean, serializer: toBooleanAttribute}) public closeOnEsc: boolean;
  /** Close the Toggleable on a click/tap outside */
  @attr({parser: parseBoolean, serializer: toBooleanAttribute}) public closeOnOutsideAction:
boolean;

  /** Initial params to pass to show/hide action on the start */
  @jsonAttr<ESLToggleableActionParams>({defaultValue: {force: true, initiator: 'init'}})
  public initialParams: ESLToggleableActionParams;
  /** Default params to merge into passed action params */
  @jsonAttr<ESLToggleableActionParams>({defaultValue: {}})
  public defaultParams: ESLToggleableActionParams;
  /** Hover params to pass from track hover listener */
  @jsonAttr<ESLToggleableActionParams>({defaultValue: {}})
  public trackHoverParams: ESLToggleableActionParams;

  /** Marker of initially opened toggleable instance */
  public initiallyOpened: boolean;

  /** Inner state */
  private _open: boolean = false;

  /** Inner show/hide task manager instance */
  protected _task: DelayedTask = new DelayedTask();
  /** Marker for current hover listener state */
  protected _trackHover: boolean = false;
  /** Delay for track hover listeners actions */
```

```
  protected _trackHoverDelay: number | undefined;

  protected override connectedCallback(): void {
    super.connectedCallback();
    if (!this.id && !this.noAutoId) {
      this.id = sequentialUID(this.baseTagName, this.baseTagName + '-');
    }
    this.initiallyOpened = this.hasAttribute('open');
    this.setInitialState();
  }

  protected override disconnectedCallback(): void {
    super.disconnectedCallback();
    activators.delete(this);
  }

  protected override attributeChangedCallback(attrName: string, oldVal: string, newVal: string): void {
    if (!this.connected || newVal === oldVal) return;
    switch (attrName) {
      case 'open': {
        const isOpen = this.hasAttribute('open');
        if (this.open === isOpen) return;
        this.toggle(isOpen, {initiator: 'attribute', showDelay: 0, hideDelay: 0});
        break;
      }
      case 'group':
        this.$$fire(this.GROUP_CHANGED_EVENT, {
          detail: {oldGroupName: oldVal, newGroupName: newVal}
        });
        break;
    }
  }

  /** Set initial state of the Toggleable */
  protected setInitialState(): void {
    if (this.initialParams) {
      this.toggle(this.initiallyOpened, this.initialParams);
```

```
    }
  }

  /** Bind outside action event listeners */
  protected bindOutsideEventTracking(track: boolean): void {
    track ? this.$$on(this._onOutsideAction) : this.$$off(this._onOutsideAction);
  }
  /** Bind hover events listeners for the Toggleable itself */
  protected bindHoverStateTracking(track: boolean, hideDelay?: number | string): void {
    if (!hasHover) return;
    this._trackHoverDelay = track && hideDelay !== undefined ? +hideDelay : undefined;
    if (this._trackHover === track) return;
    this._trackHover = track;

    track ? this.$$on(this._onMouseEnter) : this.$$off(this._onMouseEnter);
    track ? this.$$on(this._onMouseLeave) : this.$$off(this._onMouseLeave);
  }

  /** Function to merge the result action params */
  protected mergeDefaultParams(params?: ESLToggleableActionParams): ESLToggleableActionParams {
    const type = this.constructor as typeof ESLToggleable;
    return Object.assign({}, type.DEFAULT_PARAMS, this.defaultParams, copyDefinedKeys(params));
  }

  /** Toggle the element state */
  public toggle(state: boolean = !this.open, params?: ESLToggleableActionParams): ESLToggleable {
    return state ? this.show(params) : this.hide(params);
  }

  /** Change the element state to active */
  public show(params?: ESLToggleableActionParams): ESLToggleable {
    params = this.mergeDefaultParams(params);
    this._task.put(this.showTask.bind(this, params), defined(params.showDelay, params.delay));
    this.bindOutsideEventTracking(this.closeOnOutsideAction);
```

```
    this.bindHoverStateTracking(!!params.trackHover, defined(params.hideDelay, params.delay));
    return this;
  }
  /** Change the element state to inactive */
  public hide(params?: ESLToggleableActionParams): ESLToggleable {
    params = this.mergeDefaultParams(params);
    this._task.put(this.hideTask.bind(this, params), defined(params.hideDelay, params.delay));
    this.bindOutsideEventTracking(false);
    this.bindHoverStateTracking(!!params.trackHover, defined(params.hideDelay, params.delay));
    return this;
  }

  /** Actual show task to execute by toggleable task manger ({@link DelayedTask} out of the box) */
  protected showTask(params: ESLToggleableActionParams): void {
    Object.defineProperty(params, 'action', {value: 'show', writable: false});
    if (!this.shouldShow(params)) return;
    if (!params.silent && !this.$$fire(this.BEFORE_SHOW_EVENT, {detail: {params}})) return;
    this.activator = params.activator;
    this.onShow(params);
    if (!params.silent) this.$$fire(this.SHOW_EVENT, {detail: {params}, cancelable: false});
  }
  /** Actual hide task to execute by toggleable task manger ({@link DelayedTask} out of the box) */
  protected hideTask(params: ESLToggleableActionParams): void {
    Object.defineProperty(params, 'action', {value: 'hide', writable: false});
    if (!this.shouldHide(params)) return;
    if (!params.silent && !this.$$fire(this.BEFORE_HIDE_EVENT, {detail: {params}})) return;
    this.onHide(params);
    this.bindOutsideEventTracking(false);
    if (!params.silent) this.$$fire(this.HIDE_EVENT, {detail: {params}, cancelable: false});
  }

  /**
   * Actions to execute before showing of toggleable.
```

```
   * Returns false if the show action should not be executed.
   */
  protected shouldShow(params: ESLToggleableActionParams): boolean {
    return params.force || !this.open;
  }

  /**
   * Actions to execute on show toggleable.
   * Inner state and 'open' attribute are not affected and updated before `onShow` execution.
   * Adds CSS classes, update a11y and fire {@link ESLToggleable.REFRESH_EVENT} event by default.
   */
  protected onShow(params: ESLToggleableActionParams): void {
    this.open = true;
    CSSClassUtils.add(this, this.activeClass);
    CSSClassUtils.add(document.body, this.bodyClass, this);
    if (this.containerActiveClass) {
      const $container = findParent(this, this.containerActiveClassTarget);
      $container && CSSClassUtils.add($container, this.containerActiveClass, this);
    }

    this.updateA11y();
    this.$$fire(this.REFRESH_EVENT); // To notify other components about content change
  }

  /**
   * Actions to execute before hiding of toggleable.
   * Returns false if the hide action should not be executed.
   */
  protected shouldHide(params: ESLToggleableActionParams): boolean {
    return params.force || this.open;
  }

  /**
   * Actions to execute on hide toggleable.
   * Inner state and 'open' attribute are not affected and updated before `onShow` execution.
   * Removes CSS classes and update a11y by default.
   */
```

```
  protected onHide(params: ESLToggleableActionParams): void {
    this.open = false;
    CSSClassUtils.remove(this, this.activeClass);
    CSSClassUtils.remove(document.body, this.bodyClass, this);
    if (this.containerActiveClass) {
      const $container = findParent(this, this.containerActiveClassTarget);
      $container && CSSClassUtils.remove($container, this.containerActiveClass, this);
    }
    this.updateA11y();
  }

  /** Active state marker */
  public get open(): boolean {
    return this._open;
  }
  public set open(value: boolean) {
    this.toggleAttribute('open', this._open = value);
  }

  /** Last component that has activated the element. Uses {@link
ESLToggleableActionParams.activator}*/
  public get activator(): HTMLElement | null | undefined {
    return activators.get(this);
  }
  public set activator(el: HTMLElement | null | undefined) {
    el ? activators.set(this, el) : activators.delete(this);
  }

  /** Returns the element to apply a11y attributes */
  protected get $a11yTarget(): HTMLElement | null {
    const target = this.getAttribute('a11y-target');
    if (target === 'none') return null;
    return target ? this.querySelector(target) : this;
  }

  /** Called on show and on hide actions to update a11y state accordingly */
  protected updateA11y(): void {
    const targetEl = this.$a11yTarget;
```

```
  if (!targetEl) return;
  targetEl.setAttribute('aria-hidden', String(!this._open));
}

/** @returns if the passed event should trigger hide action */
public isOutsideAction(e: Event): boolean {
  const target = e.target as HTMLElement;
  // target is inside current toggleable
  if (this.contains(target)) return false;
  // target is inside last activator
  if (this.activator && this.activator.contains(target)) return false;
  // Event is not a system command key
  return !(e instanceof KeyboardEvent && SYSTEM_KEYS.includes(e.key));
}

@listen({event: 'click', selector: (el: ESLToggleable) => el.closeTrigger || ''})
protected _onCloseClick(e: DelegatedEvent<MouseEvent>): void {
  this.hide({
    initiator: 'close',
    activator: e.$delegate as HTMLElement,
    event: e
  });
}

@listen({
  auto: false,
  event: 'keydown mouseup touchend',
  target: document,
  capture: true
})
protected _onOutsideAction(e: Event): void {
  if (!this.isOutsideAction(e)) return;
  // Used 0 delay to decrease priority of the request
  this.hide({initiator: 'outsideaction', hideDelay: 0, event: e});
}

@listen('keydown')
protected _onKeyboardEvent(e: KeyboardEvent): void {
```

```
  if (this.closeOnEsc && e.key === ESC) {
    this.hide({initiator: 'keyboard', event: e});
  }
 }

 @listen({auto: false, event: 'mouseenter'})
 protected _onMouseEnter(e: MouseEvent): void {
  const baseParams: ESLToggleableActionParams = {
    initiator: 'mouseenter',
    trackHover: true,
    activator: this.activator,
    event: e,
    hideDelay: this._trackHoverDelay
  };
  this.show(Object.assign(baseParams, this.trackHoverParams));
 }
 @listen({auto: false, event: 'mouseleave'})
 protected _onMouseLeave(e: MouseEvent): void {
  const baseParams: ESLToggleableActionParams = {
    initiator: 'mouseleave',
    trackHover: true,
    activator: this.activator,
    event: e,
    hideDelay: this._trackHoverDelay
  };
  this.hide(Object.assign(baseParams, this.trackHoverParams));
 }

 /** Prepares toggle request events param */
 protected buildRequestParams(e: CustomEvent<ESLToggleableRequestDetails>):
ESLToggleableActionParams | null {
  const detail = e.detail || {};
  if (!isMatches(this, detail.match)) return null;
  return Object.assign({}, detail, {event: e});
 }

 /** Actions to execute on show request */
 @listen((el: ESLToggleable) => el.SHOW_REQUEST_EVENT)
```

```
  protected _onShowRequest(e: CustomEvent<ESLToggleableRequestDetails>): void {
    const params = this.buildRequestParams(e);
    params && this.show(params);
  }
  /** Actions to execute on hide request */
  @listen((el: ESLToggleable) => el.HIDE_REQUEST_EVENT)
  protected _onHideRequest(e: CustomEvent<ESLToggleableRequestDetails>): void {
    const params = this.buildRequestParams(e);
    params && this.hide(params);
  }
}

declare global {
  export interface ESLLibrary {
    Toggleable: typeof ESLToggleable;
  }

  export interface HTMLElementTagNameMap {
    'esl-toggleable': ESLToggleable;
  }
}
```

Додаток Д

```
import {ExportNs} from '../../esl-utils/environment/export-ns';
import {CSSClassUtils} from '../../esl-utils/dom/class';
import {attr, boolAttr, jsonAttr, listen} from '../../esl-utils/decorators';
import {afterNextRender, skipOneRender} from '../../esl-utils/async/raf';
import {ESLToggleable} from '../../esl-toggleable/core';

import type {ESLPanelGroup} from '../../esl-panel-group/core';
import type {ESLToggleableActionParams} from '../../esl-toggleable/core';

/** {@link ESLPanel} action params interface */
export interface ESLPanelActionParams extends ESLToggleableActionParams {
  /** Panel group */
  capturedBy?: ESLPanelGroup;
```

```
  /** Prevents collapsing/expanding animation */
  noAnimate?: boolean;
}

/** @deprecated alias, use {@link ESLPanelActionParams} instead. Will be removed in
v5.0.0. */
export type PanelActionParams = ESLPanelActionParams;

@ExportNs('Panel')
export class ESLPanel extends ESLToggleable {
  public static override is = 'esl-panel';

  /** Class(es) to be added for active state ('open' by default) */
  @attr({defaultValue: 'open'}) public override activeClass: string;
  /** Class(es) to be added during animation ('animate' by default) */
  @attr({defaultValue: 'animate'}) public animateClass: string;
  /** Class(es) to be added during animation after next render ('post-animate' by default) */
  @attr({defaultValue: 'post-animate'}) public postAnimateClass: string;

  /** CSS selector of the parent group (default: 'esl-panel-group') */
  @attr({defaultValue: 'esl-panel-group'}) public panelGroupSel: string;

  /** Initial params for current ESLPanel instance */
  @jsonAttr<ESLPanelActionParams>({defaultValue: {force: true, initiator: 'init'}})
  public override initialParams: ESLPanelActionParams;

  /** Active while animation in progress */
  @boolAttr({readonly: true}) public animating: boolean;

  /** Inner height state that updates after show/hide actions but before show/hide events
triggered */
  protected _initialHeight: number = 0;

  /** @returns Previous active panel height at the start of the animation */
  public get initialHeight(): number {
    return this._initialHeight;
  }
```

```
  /** @returns Closest panel group or null if not presented */
  public get $group(): ESLPanelGroup | null {
    if (this.groupName === 'none' || this.groupName) return null;
    return this.closest(this.panelGroupSel);
  }

  /** Process show action */
  protected override onShow(params: ESLPanelActionParams): void {
    this._initialHeight = this.scrollHeight;
    super.onShow(params);

    this.beforeAnimate();
    if (params.noAnimate) return this.postAnimate(params.capturedBy);
    this.onAnimate(0, this._initialHeight);
  }

  /** Process hide action */
  protected override onHide(params: ESLPanelActionParams): void {
    this._initialHeight = this.scrollHeight;
    super.onHide(params);

    this.beforeAnimate();
    if (params.noAnimate) return this.postAnimate(null);
    this.onAnimate(this._initialHeight, 0);
  }

  /** Pre-processing animation action */
  protected beforeAnimate(): void {
    this.toggleAttribute('animating', true);
    CSSClassUtils.add(this, this.animateClass);
    this.postAnimateClass && afterNextRender(() => CSSClassUtils.add(this,
this.postAnimateClass));
  }

  /** Handles post animation process to initiate after animate step */
  protected postAnimate(capturedBy?: ESLPanelGroup | null): void {
    if (capturedBy && capturedBy.animating) {
      capturedBy.$$on({
```

```
      event: capturedBy.AFTER_ANIMATE_EVENT,
      once: true
    }, () => this.afterAnimate());
  } else {
    skipOneRender(() => this.afterAnimate());
  }
}

/** Process animation */
protected onAnimate(from: number, to: number): void {
  // set initial height
  this.style.setProperty('max-height', `${from}px`);
  // make sure that browser applies initial height for animation
  afterNextRender(() => {
    this.style.setProperty('max-height', `${to}px`);
    this.fallbackAnimate();
  });
}

/** Checks if transition happens and runs afterAnimate step if transition is not presented*/
protected fallbackAnimate(): void {
  afterNextRender(() => {
    const distance = parseFloat(this.style.maxHeight) - this.clientHeight;
    if (Math.abs(distance) <= 1) this.afterAnimate();
  });
}

/** Post-processing animation action */
protected afterAnimate(): void {
  const {animating} = this;
  this.clearAnimation();
  // Prevent fallback calls from being tracked
  if (!animating) return;
  this.$$fire(this.open ? this.AFTER_SHOW_EVENT : this.AFTER_HIDE_EVENT);
}

/** Clear animation properties */
protected clearAnimation(): void {
```

```
    this.toggleAttribute('animating', false);
    this.style.removeProperty('max-height');
    CSSClassUtils.remove(this, this.animateClass);
    CSSClassUtils.remove(this, this.postAnimateClass);
  }

  /** Catching CSS transition end event to start post-animate processing */
  @listen('transitionend')
  protected _onTransitionEnd(e?: TransitionEvent): void {
    if (!e || (e.propertyName === 'max-height' && e.target === this)) {
      this.afterAnimate();
    }
  }

  /** Merge params that are used by panel group for actions */
  protected override mergeDefaultParams(params?: ESLToggleableActionParams):
ESLToggleableActionParams {
    const type = this.constructor as typeof ESLToggleable;
    const stackConfig = this.$group?.panelConfig || {};
    return Object.assign({}, stackConfig, type.DEFAULT_PARAMS, this.defaultParams,
params || {});
  }
}

declare global {
  export interface ESLLibrary {
    Panel: typeof ESLPanel;
  }
  export interface HTMLElementTagNameMap {
    'esl-panel': ESLPanel;
  }
}
```

Додаток Е

```
import {bind, attr, boolAttr, listen} from '../../../esl-utils/decorators';
import {ARROW_DOWN, ARROW_UP, ENTER, SPACE} from '../../../esl-utils/dom/keys';
import {ExportNs} from '../../../esl-utils/environment/export-ns';
```

```
import {ESLScrollbar} from '../../../esl-scrollbar/core';
import {ESLSelectItem} from './esl-select-item';
import {ESLSelectWrapper} from './esl-select-wrapper';

@ExportNs('SelectList')
export class ESLSelectList extends ESLSelectWrapper {
  public static override readonly is = 'esl-select-list';
  public static observedAttributes = ['select-all-label', 'disabled'];

  public static override register(): void {
    ESLSelectItem.register();
    super.register();
  }

  /** Select all options text */
  @attr({defaultValue: 'Select All'}) public selectAllLabel: string;

  /** Disabled state marker */
  @boolAttr() public disabled: boolean;
  /** Marker for selecting items to be pinned to the top of the list */
  @boolAttr() public pinSelected: boolean;

  protected $items: ESLSelectItem[];
  protected $list: HTMLDivElement;
  protected $scroll: ESLScrollbar;
  protected $selectAll: ESLSelectItem;

  constructor() {
    super();
    this.$list = document.createElement('div');
    this.$list.setAttribute('role', 'list');
    this.$list.classList.add('esl-scrollable-content');
    this.$list.classList.add('esl-select-list-container');
    this.$scroll = ESLScrollbar.create();
    this.$scroll.target = '::prev';
    this.$selectAll = ESLSelectItem.create();
    this.$selectAll.classList.add('esl-select-all-item');
  }
```

```
  protected override attributeChangedCallback(attrName: string, oldVal: string, newVal:
string): void {
    if (!this.connected || newVal === oldVal) return;
    if (attrName === 'select-all-label') {
      this.$selectAll.textContent = newVal;
    }
    if (attrName === 'disabled') {
      this._updateDisabled();
    }
  }

  protected override connectedCallback(): void {
    super.connectedCallback();

    this.appendChild(this.$selectAll);
    this.appendChild(this.$list);
    this.appendChild(this.$scroll);

    this.bindSelect();

    this._updateDisabled();
  }
  protected override disconnectedCallback(): void {
    super.disconnectedCallback();

    this.appendChild(this.$selectAll);
    this.appendChild(this.$list);
    this.appendChild(this.$scroll);
  }

  protected bindSelect(): void {
    const target = this.querySelector('[esl-select-target]');
    if (!target || !(target instanceof HTMLSelectElement)) return;
    this.$select = target;
  }

  protected _renderItems(): void {
```

```
    if (!this.$select) return;
    this.$list.innerHTML = '';
    this.$items = this.options.map(ESLSelectItem.build);
    if (this.pinSelected) {
      this._renderGroup(this.$items.filter((option) => option.selected));
      this._renderGroup(this.$items.filter((option) => !option.selected));
    } else {
      this._renderGroup(this.$items);
    }
    this.toggleAttribute('multiple', this.multiple);
  }
  protected _renderGroup(items: ESLSelectItem[]): void {
    items.forEach((item) => this.$list.appendChild(item));
    const [last] = items.slice(-1);
    last && last.classList.add('last-in-group');
  }

  protected _updateSelectAll(): void {
    if (!this.multiple) {
      this.$selectAll.removeAttribute('tabindex');
      return;
    }
    this.$selectAll.tabIndex = 0;
    this.$selectAll.selected = this.isAllSelected();
    this.$selectAll.textContent = this.selectAllLabel;
  }
  protected _updateDisabled(): void {
    this.setAttribute('aria-disabled', String(this.disabled));
    if (!this.$select) return;
    this.$select.disabled = this.disabled;
  }

  @bind
  protected override _onTargetChange(newTarget: HTMLSelectElement | undefined,
oldTarget: HTMLSelectElement | undefined): void {
    super._onTargetChange(newTarget, oldTarget);
    this._updateSelectAll();
    this._renderItems();
```

```
}

@listen({inherit: true})
public override _onChange(): void {
  this._updateSelectAll();
  this.$items.forEach((item) => item.update());
}

@bind
public override _onListChange(): void {
  this._renderItems();
}

@listen('click')
protected _onClick(e: MouseEvent | KeyboardEvent): void {
  if (this.disabled) return;
  const target = e.target;
  if (!target || !(target instanceof ESLSelectItem)) return;
  if (target.classList.contains('esl-select-all-item')) {
    this.setAllSelected(!target.selected);
  } else {
    this.setSelected(target.value, !target.selected);
  }
}

@listen('keydown')
protected _onKeydown(e: KeyboardEvent): void {
  if ([ENTER, SPACE].includes(e.key)) {
    this._onClick(e);
    e.preventDefault();
  }
  if ([ARROW_UP, ARROW_DOWN].includes(e.key)) {
    const index = this.$items.indexOf(document.activeElement as ESLSelectItem);
    const count = this.$items.length;
    const increment = e.key === ARROW_UP ? -1 : 1;
    if (index === -1) return;
    this.$items[(index + increment + count) % count].focus();
    e.preventDefault();
```

```
    }
  }
}

declare global {
  export interface ESLLibrary {
    SelectList: typeof ESLSelectList;
  }
  export interface HTMLElementTagNameMap {
    'esl-select-list': ESLSelectList;
  }
}
```

Додаток Є

```
import {ExportNs} from '../../esl-utils/environment/export-ns';
import {ready, memoize, attr, boolAttr} from '../../esl-utils/decorators';
import {ESLTraversingQuery} from '../../esl-traversing-query/core';
import {parseNumber} from '../../esl-utils/misc/format';
import {ESLBaseElement} from '../../esl-base-element/core';

import {ESLAnimateService} from './esl-animate-service';

/**
 * ESLAnimate - custom element for quick {@link ESLAnimateService} attaching
 *
 * Have two types of usage:
 * - trough the target-s definition, then esl-animate (without it's own content) became invisible plugin-component
 * `<esl-animate target="::next"></esl-animate><div>Content</div>`
 * - trough the content wrapping
 * `<esl-animate>Content</esl-animate>`
 */
@ExportNs('Animate')
export class ESLAnimate extends ESLBaseElement {
  public static override is = 'esl-animate';
  public static observedAttributes = ['group', 'repeat', 'target'];
```

```
  /**
   * Class(es) to add on viewport intersection
   * @see ESLAnimateConfig.cls
   */
  @attr({defaultValue: 'in'}) public cls: string;

  /**
   * Enable group animation for targets
   * @see ESLAnimateConfig.group
   */
  @boolAttr() public group: boolean;

  /**
   * Delay to start animation from previous item in group
   * @see ESLAnimateConfig.groupDelay
   */
  @attr({defaultValue: '100'}) public groupDelay: string;

  /**
   * Re-animate item after its getting hidden
   * @see ESLAnimateConfig.repeat
   */
  @boolAttr() public repeat: boolean;

  /**
   * Intersection ratio to consider element as visible.
   * Only 0.2 (20%), 0.4 (40%), 0.6 (60%), 0.8 (80%) values are allowed due to share of
IntersectionObserver instance
   * with a fixed set of thresholds defined.
   */
  @attr() public ratio: string;

  /**
   * Define target(s) to observe and animate
   * Uses {@link ESLTraversingQuery} with multiple targets support
   * Default: `` - current element, `<esl-animate>` behave as a wrapper
   */
  @attr() public target: string;
```

```
  /** Elements-targets found by target query */
  @memoize()
  public get $targets(): HTMLElement[] {
    return ESLTraversingQuery.all(this.target, this) as HTMLElement[];
  }

  protected override attributeChangedCallback(): void {
    if (!this.connected) return;
    this.reanimate();
  }

  @ready
  protected override connectedCallback(): void {
    super.connectedCallback();
    this.reanimate();
  }

  @ready
  protected override disconnectedCallback(): void {
    super.disconnectedCallback();
    ESLAnimateService.unobserve(this.$targets);
  }

  /** Reinitialize {@link ESLAnimateService} for targets */
  public reanimate(): void {
    ESLAnimateService.unobserve(this.$targets);
    memoize.clear(this, '$targets');
    ESLAnimateService.observe(this.$targets, {
      force: true,
      cls: this.cls,
      ratio: parseNumber(this.ratio, 0.4),
      repeat: this.repeat,
      group: this.group,
      groupDelay: parseNumber(this.groupDelay, 0)
    });
  }
}
```

```
declare global {
  export interface ESLLibrary {
    Animate: typeof ESLAnimate;
  }
  export interface HTMLElementTagNameMap {
    'esl-animate': ESLAnimate;
  }
}
```

Додаток Ж

```
import {ExportNs} from '../../esl-utils/environment/export-ns';
import {ESLBaseElement} from '../../esl-base-element/core';
import {attr, boolAttr, ready, decorate, listen, memoize} from '../../esl-utils/decorators';
import {isMatches} from '../../esl-utils/dom/traversing';
import {microtask} from '../../esl-utils/async';
import {parseBoolean, parseTime, sequentialUID} from '../../esl-utils/misc';

import {CSSClassUtils} from '../../esl-utils/dom/class';
import {ESLTraversingQuery} from '../../esl-traversing-query/core';
import {ESLMediaRuleList} from '../../esl-media-query/core';
import {ESLResizeObserverTarget} from '../../esl-event-listener/core';

import {normalize, toIndex, canNavigate} from './esl-carousel.utils';

import {ESLCarouselSlide} from './esl-carousel.slide';
import {ESLCarouselRenderer} from './esl-carousel.renderer';
import {ESLCarouselChangeEvent} from './esl-carousel.events';

import type {
  ESLCarouselState,
  ESLCarouselSlideTarget,
  ESLCarouselStaticState,
  ESLCarouselConfig,
  ESLCarouselActionParams
} from './esl-carousel.types';

@ExportNs('Carousel')
```

```
export class ESLCarousel extends ESLBaseElement {
  public static override is = 'esl-carousel';
  public static observedAttributes = ['media', 'type', 'loop', 'count', 'vertical', 'step-duration',
'container'];

  /** Media query pattern used for {@link ESLMediaRuleList} of `type`, `loop` and `count`
(default: `all`) */
  @attr({defaultValue: 'all'}) public media: string;
  /** Renderer type name (`multi` by default). Supports {@link ESLMediaRuleList} syntax */
  @attr({defaultValue: 'default'}) public type: string;
  /** Marker to enable loop mode for a carousel (`true` by default). Supports {@link
ESLMediaRuleList} syntax */
  @attr({defaultValue: 'false'}) public loop: string | boolean;
  /** Count of slides to show on the screen (`1` by default). Supports {@link
ESLMediaRuleList} syntax */
  @attr({defaultValue: '1'}) public count: string | number;
  /** Orientation of the carousel (`horizontal` by default). Supports {@link
ESLMediaRuleList} syntax */
  @attr({defaultValue: 'false'}) public vertical: string | boolean;

  /** Duration of the single slide transition */
  @attr({defaultValue: '250'}) public stepDuration: string;

  /** Container selector (supports traversing query). Carousel itself by default */
  @attr({defaultValue: ''}) public container: string;
  /** CSS class to add on the container when carousel is empty */
  @attr({defaultValue: ''}) public containerEmptyClass: string;
  /** CSS class to add on the container when carousel is incomplete */
  @attr({defaultValue: ''}) public containerIncompleteClass: string;

  /** true if carousel is in process of animating */
  @boolAttr({readonly: true}) public animating: boolean;
  /** true if carousel is empty */
  @boolAttr({readonly: true}) public empty: boolean;
  /** true if carousel has only one item */
  @boolAttr({readonly: true}) public singleSlide: boolean;
  /** true if carousel is incomplete (total slides count is less or equal to visible slides count) */
  @boolAttr({readonly: true}) public incomplete: boolean;
```

```
/** Marker/mixin attribute to define slide element */
public get slideAttrName(): string {
  return this.tagName + '-slide';
}

/** Renderer type {@link ESLMediaRuleList} instance */
@memoize()
public get typeRule(): ESLMediaRuleList<string> {
  return ESLMediaRuleList.parse(this.type, this.media);
}
/** Loop marker {@link ESLMediaRuleList} instance */
@memoize()
public get loopRule(): ESLMediaRuleList<boolean> {
  return ESLMediaRuleList.parse(this.loop as string, this.media, parseBoolean);
}
/** Count of visible slides {@link ESLMediaRuleList} instance */
@memoize()
public get countRule(): ESLMediaRuleList<number> {
  return ESLMediaRuleList.parse(this.count as string, this.media, parseInt);
}
/** Orientation of the carousel {@link ESLMediaRuleList} instance */
@memoize()
public get verticalRule(): ESLMediaRuleList<boolean> {
  return ESLMediaRuleList.parse(this.vertical as string, this.media, parseBoolean);
}
/** Duration of the single slide transition {@link ESLMediaRuleList} instance */
@memoize()
public get stepDurationRule(): ESLMediaRuleList<number> {
  return ESLMediaRuleList.parse(this.stepDuration, this.media, parseTime);
}

/** Returns observed media rules */
public get observedRules(): ESLMediaRuleList[] {
  return [this.typeRule, this.loopRule, this.countRule, this.verticalRule];
}

/** Carousel instance current {@link ESLCarouselStaticState} */
public get config(): ESLCarouselStaticState {
```

```
    return this.renderer.config;
  }

  /** Carousel instance configured {@link ESLCarouselStaticState} */
  public get configCurrent(): ESLCarouselConfig {
    return {
      type: this.typeRule.value || 'default',
      size: this.$slides.length,
      count: this.countRule.value || 1,
      loop: !!this.loopRule.value,
      vertical: !!this.verticalRule.value
    };
  }

  /** Carousel instance current {@link ESLCarouselState} */
  public get state(): ESLCarouselState {
    return Object.assign({}, this.renderer.config, {
      activeIndex: this.activeIndex
    });
  }

  /** @returns currently active renderer */
  @memoize()
  public get renderer(): ESLCarouselRenderer {
    return ESLCarouselRenderer.registry.create(this, this.configCurrent);
  }

  @ready
  protected override connectedCallback(): void {
    super.connectedCallback();
    this.update();
    this.updateA11y();
  }

  protected override attributeChangedCallback(attrName: string, oldVal: string, newVal:
string): void {
    if (!this.connected) return;
    if (attrName === 'container') {
```

```
      memoize.clear(this, '$container');
      return this.updateStateMarkers();
    }
    memoize.clear(this, `${attrName}Rule`);
    this.update();
  }

  protected override disconnectedCallback(): void {
    super.disconnectedCallback();
    memoize.clear(this, ['$container', '$slides', '$slidesArea', 'typeRule', 'loopRule', 'countRule',
'verticalRule']);
  }

  /** Updates the config and the state that is associated with */
  @decorate(microtask)
  public update(): void {
    const config = this.configCurrent;
    const oldConfig = this.config;
    const initial = !this.renderer.bound;
    const $oldSlides = initial ? [] : this.$slides;

    memoize.clear(this, '$slides');
    const added = this.$slides.filter((slide) => !$oldSlides.includes(slide));
    const removed = $oldSlides.filter((slide) => !this.$slides.includes(slide));

    if (!added.length && !removed.length && this.renderer.equal(config)) return;

    this.renderer.unbind();
    memoize.clear(this, 'renderer');
    this.renderer.bind();

    this.updateStateMarkers();
    this.dispatchEvent(ESLCarouselChangeEvent.create({initial, added, removed, config,
oldConfig}));
  }

  protected updateStateMarkers(): void {
    this.$$attr('empty', !this.size);
```

```
  this.$$attr('single-slide', this.size === 1);
  this.$$attr('incomplete', this.size <= this.renderer.count);

  if (!this.$container) return;
  CSSClassUtils.toggle(this.$container, this.containerEmptyClass, this.empty, this);
  CSSClassUtils.toggle(this.$container, this.containerIncompleteClass, this.incomplete, this);
}

/** Appends slide instance to the current carousel */
public addSlide(slide: HTMLElement): void {
  slide.setAttribute(this.slideAttrName, '');
  if (slide.parentNode === this.$slidesArea) return this.update();
  console.debug('[ESL]: ESLCarousel moves slide to correct location', slide);
  this.$slidesArea.appendChild(slide);
}

/** Remove slide instance from the current carousel */
public removeSlide(slide: HTMLElement): void {
  if (slide.parentNode === this.$slidesArea) this.$slidesArea.removeChild(slide);
  if (this.$slides.includes(slide)) this.update();
}

protected updateA11y(): void {
  if (!this.role) {
    this.setAttribute('role', 'region');
    this.setAttribute('aria-roledescription', 'Carousel');
  }
  if (!this.id) this.id = sequentialUID('esl-carousel-');
  if (!this.$slidesArea.id) this.$slidesArea.id = `${this.id}-slides`;
  if (!this.$slidesArea.role) this.$slidesArea.role = 'list';
}

@listen({event: 'change', target: ($this: ESLCarousel) => $this.observedRules})
protected _onRuleUpdate(): void {
  this.update();
}

@listen({event: 'change', target: ESLCarouselRenderer.registry})
```

```
protected _onRegistryUpdate(): void {
  this.update();
}

@listen({event: 'resize', target: ESLResizeObserverTarget.for})
protected _onResize(): void {
  this.renderer && this.renderer.redraw();
}

@listen('esl:show:request')
protected onShowRequest(e: CustomEvent): void {
  const detail = e.detail || {};
  if (!isMatches(this, detail.match)) return;
  const index = this.$slides.findIndex(($slide) => $slide.contains(e.target as Element));
  if (index !== -1 && !this.isActive(index)) this.goTo(index);
}

/** @returns slides that are processed by the current carousel. */
@memoize()
public get $slides(): HTMLElement[] {
  const {slideAttrName} = this;
  const els = this.$slidesArea ? [...this.$slidesArea.children] as HTMLElement[] : [];
  return els.filter((el) => el.hasAttribute(slideAttrName));
}

/**
 * @returns carousel container
 */
@memoize()
public get $container(): Element | null {
  return ESLTraversingQuery.first(this.container, this) as HTMLElement;
}

/** @returns carousel slides area */
@memoize()
public get $slidesArea(): HTMLElement {
  const $provided = this.querySelector(`[${this.tagName}-slides]`);
  if ($provided) return $provided as HTMLElement;
```

```
  const $container = document.createElement('div');
  $container.setAttribute(this.tagName + '-slides', '');
  this.appendChild($container);
  return $container;
}

/** @returns first active slide */
public get $activeSlide(): HTMLElement | undefined {
  return this.$slides[this.activeIndex];
}

/** @returns list of active slides. */
public get $activeSlides(): HTMLElement[] {
  return this.activeIndexes.map((index) => this.$slides[index]);
}

/** @returns count of slides. */
public get size(): number {
  return this.$slides.length || 0;
}

/** @returns index of first (the most left in the loop) active slide */
public get activeIndex(): number {
  if (this.size <= 0) return -1;
  if (this.isActive(0)) {
    for (let i = this.size - 1; i > 0; --i) {
      if (!this.isActive(i)) return normalize(i + 1, this.size);
    }
  }
  return this.$slides.findIndex(this.isActive, this);
}

/** @returns list of active slide indexes. */
public get activeIndexes(): number[] {
  const start = this.activeIndex;
  if (start < 0) return [];
  const indexes = [];
  for (let i = 0; i < this.size; i++) {
```

```
    const index = normalize(i + start, this.size);
    if (this.isActive(index)) indexes.push(index);
   }
   return indexes;
  }

  /** Goes to the target according to passed params */
  public goTo(target: HTMLElement | ESLCarouselSlideTarget, params:
Partial<ESLCarouselActionParams> = {}): Promise<void> {
   if (target instanceof HTMLElement) return this.goTo(this.indexOf(target), params);
   if (!this.renderer) return Promise.reject();
   return this.renderer.navigate(toIndex(target, this.state), this.mergeParams(params));
  }

  /** Moves slides by the passed offset */
  public move(offset: number, from: number = this.activeIndex, params:
Partial<ESLCarouselActionParams> = {}): void {
   if (!this.renderer) return;
   this.renderer.move(offset, from, this.mergeParams(params));
  }

  /** Commits slides to the nearest stable position */
  public commit(offset: number, from: number = this.activeIndex, params:
Partial<ESLCarouselActionParams> = {}): Promise<void> {
   if (!this.renderer) return Promise.reject();
   return this.renderer.commit(offset, from, this.mergeParams(params));
  }

  /** Merges request params with default params */
  protected mergeParams(params: Partial<ESLCarouselActionParams>):
ESLCarouselActionParams {
   const stepDuration = this.stepDurationRule.value || 0;
   return {stepDuration, ...params};
  }

  /** @returns slide by index (supports not normalized indexes) */
  public slideAt(index: number): HTMLElement {
   return this.$slides[normalize(index, this.$slides.length)];
  }
```

```
/** @returns index of the passed slide */
public indexOf(slide: HTMLElement): number {
  return this.$slides.indexOf(slide);
}

/** @returns if the passed slide target can be reached */
public canNavigate(target: ESLCarouselSlideTarget): boolean {
  return canNavigate(target, this.state);
}

/** @returns if the passed element (or slide on a passed index) is an active slide */
public isActive(el: number | HTMLElement): boolean {
  if (typeof el === 'number') return this.isActive(this.$slides[el]);
  return el && el.hasAttribute('active');
}
/** @returns if the passed element (or slide on a passed index) is a slide in pre-active state */
public isPreActive(el: number | HTMLElement): boolean {
  if (typeof el === 'number') return this.isPreActive(this.$slides[el]);
  return el && el.hasAttribute('pre-active');
}
/** @returns if the passed element (or slide on a passed index) is a next slide */
public isNext(el: number | HTMLElement): boolean {
  if (typeof el === 'number') return this.isNext(this.$slides[el]);
  return el && el.hasAttribute('next');
}
/** @returns if the passed element (or slide on a passed index) is a prev slide */
public isPrev(el: number | HTMLElement): boolean {
  if (typeof el === 'number') return this.isPrev(this.$slides[el]);
  return el && el.hasAttribute('prev');
}

/**
 * Registers component in the {@link customElements} registry
 * @param tagName - custom tag name to register custom element
 */
public static override register(tagName?: string): void {
  super.register(tagName);
```

```
    ESLCarouselSlide.is = this.is + '-slide';
    ESLCarouselSlide.register();
  }
}

declare global {
  export interface ESLCarouselNS {
  }

  export interface ESLLibrary {
    Carousel: typeof ESLCarousel & ESLCarouselNS;
  }

  export interface HTMLElementTagNameMap {
    'esl-carousel': ESLCarousel;
  }
}
```