

**ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ**  
**Навчально-науковий інститут економіки, управління, права та**  
**інформаційних технологій**  
**Кафедра інформаційних систем та технологій**

## **КВАЛІФІКАЦІЙНА РОБОТА**

на здобуття ступеня вищої освіти магістр

на тему: **«Методи оптимізації програмного коду вебдодатків на етапі розробки із застосуванням мовних моделей штучного інтелекту»**

Виконала: здобувачка вищої освіти  
за освітньою програмою  
Інформаційні управляючі системи  
та технології  
спеціальності 126 Інформаційні  
системи та технології  
ступеня вищої освіти магістр  
групи 126ІСТ\_мз\_2023[1](л.н.)  
Скриль Олена Василівна  
Керівник: Копішинська Олена  
Петрівна  
Рецензент: Муравльов Володимир  
В'ячеславович

Полтава – 2024 року

## ВСТУП

*Актуальність* теми кваліфікаційної роботи пов'язана з питаннями оптимізації та раціоналізації програмного коду на етапі розроблення вебдодатків, важливість та функціональність яких стрімко розширюється. Постійна «гонка озброєнь» в галузі розробки вебдодатків призводить до появи величезної кількості інструментів. Розуміння та впровадження в роботу новітніх технологій, особливо доступних моделей штучного інтелекту, стає критично важливим для досягнення конкурентних переваг. Як відомо, вебдодатки широко використовуються в різних сферах, забезпечують інтерфейс клієнтської частини інформаційних систем із розподіленою архітектурою, що підкреслює необхідність дослідження для вдосконалення функціональності та швидкодії вебдодатків у всіх виробничих, наукових та бізнес-аспектах.

Враховуючи широкий спектр можливостей, які надають програмні засоби веброботи, важливо провести ретельне дослідження їх характеристик, особливостей раціонального вибору, використання і оптимізації, а також з'ясувати області перетину з функціями штучного інтелекту.

*Зв'язок роботи з науковими програмами, темами.* Робота виконана у відповідності до науково-дослідної ініціативної теми «Організаційно-методологічні аспекти впровадження інформаційно-комунікаційних систем і технологій в управлінні діяльністю сучасних організацій та підприємств за умов переходу до цифрової економіки» ДРН 0123U105060, яка реалізується на кафедрі інформаційних систем та технологій Полтавського державного аграрного університету (ПДАУ).

*Метою* кваліфікаційної роботи є теоретичне та експериментальне дослідження ефективності використання різних фреймворків мови програмування JavaScript та можливостей сучасних версій великих мовних моделей штучного інтелекту для технічної оптимізації коду вебдодаткі на

етапах розробки та верифікації. Для досягнення мети були визначені та реалізовані такі наукові завдання:

- дослідити тенденції розвитку сучасних вебтехнологій та їх вплив на підходи до розробки вебдодатків;
- провести різнопланове дослідження продуктивності вебфреймворків в різних тестах та ефективності фреймворків відомих мов вебпрограмування;
- провести аналіз вимог щодо опимальності та ункціональності написання кодів у вебдодатках;
- провести експериментальне дослідження та узагальнити результати щодо використання можливостей великих мовних моелей штучного інтелекту для удосконалення й оптимізації розробленого коду вебдодатку.

*Об'єктом дослідження* кваліфікаційної роботи є заходи технічної та семантичної оптимізації програмного коду вебдодатків із використанням спеціалізованих вебтехнологій та мовних моделей штучного інтелекту на етапах розробки вебдодатків.

*Предметом дослідження* є: порівняльні властивості й технічні аспекти застосування фреймворків мови програмування JavaScript, найбільш відомих мовних моделей штучного інтелекту, а також методології формування запитів до штучного інтелекту в процесі роботи з програмним кодом.

*Методи наукових досліджень*: абстрактно–логічний, інформаційно–пошуковий, аналітичний, економіко–статистичний, методи тестування framework-benchmark та Chrome DevTools, великі закриті мовні моделі штучного інтелекту.

*Інформаційну базу* кваліфікаційної роботи складають: наукові статті, тематична література, статистичні дані аналітичних компаній, які розміщені на вебсайтах у вільному доступі, власні спостереження та фахові літературні дослідження з досліджуваної проблеми, а також відкриті моделі штучного інтелекту різних розробників, що доступні через мережу інтернет.

*Елементи наукової новизни* полягають у здійсненні нетривіального аналізу можливостей штучного інтелекту для оптимізації та підвищення

функціональності коду вебдодатків, отриманні оціночних показників ефективності відомих фреймворків, тестуванні спроможностей моделей штучного інтелекту до розпізнання й оптимізації програмного коду та висвітленні неоднозначності вибору та ефективності кожного з них.

*Практична значущість одержаних результатів:* результати роботи являють собою оригінальну працю з можливістю практичного використання та сформованих напрямків подальших наукових досліджень. Застосування результатів кваліфікаційної роботи дає змогу розширити можливості технологій розробки вебдодатків та їхнього вдосконалення на етапах тестування й реалізації. Залучення до експерименту моделей штучного інтелекту дозволяє прогнозувати в недалекому майбутньому перехід на абсолютно нові методи роботи з програмним кодом, надання розробникам нових знань та перспектив.

*Апробація результатів* дослідження відбувалася шляхом оприлюднення доповідей на наукових міжнародних конференціях із публікацією результатів у вигляді статей у збірниках матеріалів конференцій.

*Публікації.* За результатами проведеного дослідження опубліковано наукову статтю «Features of using frameworks and artificial intelligence models for javascript code optimization in modern web application development» у співавторстві з науковим керівником, яка подана на індексацію в науково-метричних базах Scopus та Google Academia.

*Структура та обсяг кваліфікаційної роботи.* Пояснювальна записка кваліфікаційної роботи складається зі вступу, 3 розділів, висновків, списку використаних джерел (65 найменувань), 4 додатків. Кваліфікаційна робота містить 6 таблиць, 17 рисунків, викладена на 60 сторінках.

# РОЗДІЛ 1

## ПРОГРАМНО–ТЕХНІЧНІ ЗАСОБИ РАЦІОНАЛІЗАЦІЇ РОЗРОБКИ ПРОГРАМНОГО КОДУ ВЕБДОДАТКІВ

### 1.1 Сучасні тенденції розвитку базових технологій і мов програмування для розробки вебдодатків

З моменту створення і до наших днів інтернет зазнав значних змін. Спершу були статичні HTML–сторінки (Web 1.0). На зміну Web 1.0 прийшов Web 2.0, а разом із ним динамічні сайти та змога взаємодіяти з користувачами, що дало великий простір для реалізації ідей розробників. Наразі на підході третя ітерація – Web 3.0, яка базуватиметься на децентралізації, використанні штучного інтелекту, одноранговій системі комунікацій та обміну інформацією з використанням технології blockchain [1-2].

При розробці сучасного вебдодатку розрізняють два поняття: frontend– та backend–розробка. Вони тісно пов’язані між собою, але це зовсім різні напрями програмування як по типу задач, що вони вирішують, так і по інструментарію, що використовується. Під поняттям frontend мається на увазі видима для користувача частина застосунку (не обов’язково сайту), а під поняттям backend – все те, що приховано від користувача. Важливо розуміти, що для обох складових вебдодатку існує велика кількість різноманітних інструментів розробки і однією із задач сучасного веброзробника є підбір найкращого набору інструментів саме для його потреб.

Обов’язковим для реалізації frontend–частини сайту є лише HTML. Мова розмітки гіпертексту HTML (Hypertext Markup Language) – основа будь якого сайту, його каркас. За допомогою нього описується сама структура сайту, елементи які на ньому будуть знаходитися [3]. Все інше, включаючи CSS, JavaScript та різні бібліотеки, фреймворки й препроцесори не є обов’язковим з точки зору браузера, але в сучасному світі без подібних інструментів не створюється жоден вебдодаток.

Оскільки HTML дозволяє лише розміщувати елементи на сторінці, постає необхідність в інструменті, який буде вказувати браузеру як ці об'єкти виглядають. Рішенням цієї проблеми є каскадні таблиці стилів CSS (Cascading Style Sheets). За допомогою таблиці стилів можна задавати, розмір, колір, положення одних елементів відносно інших, поведінку курсора при наведенні на елемент та багато інших. Препроцесори дозволяють розширити функціонал HTML при роботі з верстанням. Написаний за допомогою препроцесора код після інтерпретації сервером перетворюється в звичайний HTML. Прикладом таких препроцесорів може бути Pug або Haml.

До переваг використання HTML–препроцесорів відноситься:

- простіша робота з великим об'ємом коду;
- дотримання принципу DRY (Don't Repeat Yourself);
- зменшення загального часу необхідного на розробку та підтримка організації коду [4].

Популярність технологій розробки вебдодатків різної складності та призначення із застосуванням мов програмування Python, JavaScript, Ruby, PHP та інших у поєднанні з базовими технологіями HTML&CSS стрімко зростає, що підтверджується щорічними рейтингами мов програмування відомих аналітичних груп [5]. Різноманітність завдань, еволюція технологій веб-розробки та зростаючі вимоги до продуктивності, безпеки, масштабованості веб-додатків призвели до значної диверсифікації інструментарію розробника, а також появи засобів спеціалізації та спрощення при створенні окремих проєктів. Наприклад, основною мовою для розробки динамічного інтерфейсу користувача є мова JavaScript, яка розроблялася та розвивалася від початку запуску перших вебсайтів [6].

JavaScript – мова програмування, яка використовується веброзробниками для створення динамічних і інтерактивних елементів сайту. Існування JavaScript зробило можливим існування вебдодатків – застосунків, в яких оновлення інформації відбувається без оновлення всієї сторінки [7]. У JavaScript фактично немає конкурентів: 98% вебсайтів використовують саме

цю мову. Але альтернативи все ж є: Dart, CoffeeScript, Haxe, ClojureScript. Вважати їх повноцінними аналогами не можна, адже в результаті всі вони все рівно компілюються в JavaScript код. Рейтинг мов програмування за версією IEEE Spectrum представлено на рис. 1.1 [5].

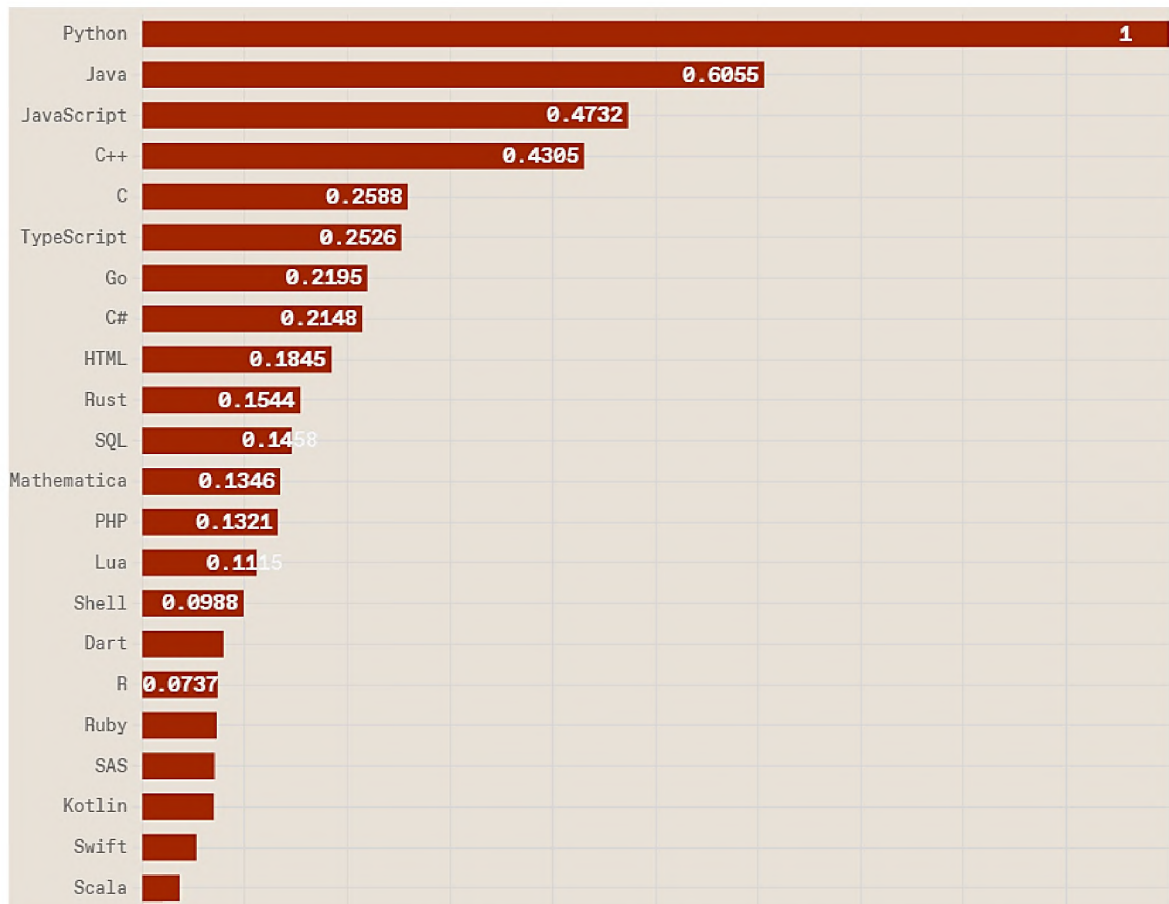


Рисунок 1.1 – Позиція JavaScript у рейтингу мов програмування за даними [5]

Окремо варто розглядати TypeScript – мову програмування, що розширює JavaScript шляхом надання можливості статичної типізації (знаходиться в топі рейтингу мов вебпрограмування, див. рис. 1.1). TypeScript виник як надмножина JavaScript, яка додає статичну типізацію, що підвищує стабільність і зручність розробки великих проектів [8]. Використання TypeScript зазвичай не є обов'язковим (за винятком деяких фреймворків, наприклад – Angular), але його використання дуже бажано, особливо в проектах з великою кількістю класів, функцій та змінних. Розробнику, який використовує TypeScript не потрібно витратити час на те щоб дізнатися, які

поля є у об'єкта, або що повертає та чи інша функція. В цьому йому допоможе IDE, яка повністю підтримує TypeScript, та буде показувати розробнику всю необхідну інформацію своєю функцією autocomplete (яка дуже погано працює з звичайним JavaScript) [9].

Нові мови часто розробляються для полегшення специфічних задач або для надання більш зручних інструментів для певних типів проєктів. Наприклад, Elm і Svelte були створені, щоб зробити розробку фронтенду більш передбачуваною та простою, з мінімальними помилками та високою продуктивністю. Різні компанії, організації та розробники експериментують з новими підходами до розробки веб-додатків. Це породжує конкуренцію та прагнення створити кращу мову програмування або фреймворк, що відповідає конкретним вимогам. На популярність мов програмування значний вплив мають також великі активні спільноти, що сприяють їх розвитку, створенню фреймворків та бібліотек. Наприклад, JavaScript став основною мовою для веброзробки завдяки широкій екосистемі бібліотек React, Angular, Vue.js, а також фреймворків.

Однак, на тлі еволюції та урізноманітнення технологій із стрімким зростанням кількості вебдодатків стали виявлятися й нові проблеми, які раніше не були такими очевидними. Наприклад, уніфікація та раціоналізація коду: кожен із представників масової професії веброзробника пише код у власному стилі. Раніше у кожній вебстудії, або навіть кожного окремого розробника могла бути власна реалізація таких базових речей, як система авторизації, підключення та спілкування з базою даних тощо. Коли до проєкту долучається новий розробник, то йому потрібно довгий час вивчати підхід до написання коду, який використовувався попередніми прогамістами. Внаслідок такого підходу утворилась велика кількість «spaghetti code», який веб-розробникам до цього часу доводиться розбирати й переписувати на сучасний лад. Іншою проблемою стала велика кількість коду, який безпосередньо треба писати. Частковим рішенням цих проблем стали бібліотеки та фреймворки.

## 1.2 Призначення і особливості застосування фреймворків мов програмування в розробці вебдодатків

Мова програмування JavaScript є інтегратором функцій сайту, керує його інтерактивністю та робить більш сприйнятним до дій користувача, дозволяє оновлювати зміст сайту залежно від запитів відвідувача. Сучасні сайти можливо розробляти й без використання фреймворків, але втім саме вони значно пришвидшують швидкість розробки і додають купу нових можливостей для розробника. Професіонали, які повсякчас стикаються з величезною кількістю рутинних або повторюваних операцій, в першу чергу оцінили інноваційні підходи до побудови кодів, які надали бібліотеки та фреймворки.

Фреймворки і бібліотеки містять код, який написав інший розробник. Цей код використовують, коли вирішують однотипні завдання чи поширені проблеми. Принцип раціональної розробки програмного забезпечення передбачає не дублювати код, який багато разів використовується. Фреймворк можна розглядати як певний набір інструментів, стандартів, підходів до написання коду. Саме він керує розробником та тим, як буде написаний код.

Фреймворки часто перетинаються з бібліотеками. Розробники використовують обидва компоненти, і новачки часто не розуміють, чим вони відрізняються. Стислі визначення вирізняють сутність двох понять. Фреймворк це платформа для створення вебсайтів та програм, яка полегшує розробку завдяки великій кількості реалізованих функцій. Програмістові не треба писати код із нуля. Він бере готове рішення і створює надбудову для реалізації продукту [10].

Бібліотеки – готові окремі компоненти, які вирішують певні завдання [10]. Наприклад, є бібліотеки для обробки файлів та виведення картинки на екран. За принципом дії відмінність полягає в наступному: фреймворк звертається до додатку, який містить основний код, та реалізує в ньому певні дії; до бібліотеки ж, навпаки, звертається додаток і включає її готові елементи,

поєднуючи із основним планом, проєктом. Основна відмінність фреймворку від бібліотеки пояснюється тим, що фреймворк ніби задає жорсткі рамки. Розробник інтегрує свій код у стороннє рішення, але не може вийти за межі стандартної логіки. Бібліотеки ж можна або використовувати будь-якої миті, або відключити зовсім, якщо є альтернативи [11]. Бібліотека – комплекс модулів, функцій та компонентів та іншого «готового» коду. По суті, це збірник коду, який можна повторно використовувати в своїх проєктах [12]. Завдяки ним веброзробнику не потрібно писати з нуля такі речі як наприклад модальні вікна, каруселі зображень, слайдери і багато інших. Це значно спрощує процес розробки.

Вказані відмінності бажано розуміти, щоб правильно використовувати обидві технології. Коли замовник просить створити сайт на базі готової CMS, PHP-фреймворки вже не знадобляться, а ось CSS можуть стати в нагоді. Можна підключити їх до проєкту та не писати код стандартних функцій. У бібліотек та фреймворків спільна мета – звільнити програміста від вирішення однотипних завдань.

Фреймворк обмежує розробника, не даючи вийти за свої рамки. Додатки написані на одному фреймворкові будуть мати майже однакову структуру файлів, ідентичні патерни розробки та підхід до вирішення задач. Окрім цього, більшість сучасних фреймворків мають великий арсенал вбудованого функціоналу, що вберігає розробників від «винайдення велосипедів», а також надає розробникам можливість сфокусуватися на більш важливих аспектах розробки, наприклад на бізнес-логіці [13].

З часом кількість готових рішень стала збільшуватись зі значною швидкістю. Існує жарт, що якщо відкрити словник англійської мови і обрати звідти будь-яке слово, то з такою назвою вже буде існувати JavaScript фреймворк. Кожен фреймворк має свої особливості, переваги та недоліки. Деякі з них краще підходять для розробки маленьких і простих проєктів, а деякі розроблені спеціально для використання в масштабних проєктах, які пишуться не один рік.

Існують PHP, CSS та JS-фреймворки. Кожен вид програмних інструментів вирішує свої завдання, але вони об'єднані спільною метою – допомогти розробнику позбутися рутини. Замість писати, наприклад, систему виведення шаблонів сторінки кожного проекту, програміст може займатися нестандартними завданнями. Фреймворки та бібліотеки не є легким вирішенням проблем. Вони лише пропонують «фундамент» з урахуванням якого можна створити проект. Якщо розробник сподівається, що з нього зроблять всю роботу, ці інструменти не вміють.

PHP-фреймворки використовуються на бекенд для формування базової структури. Для фронтенду існує велика кількість своїх фреймворків, які допомагають швидко зібрати каркас і змінити його зовнішній вигляд.

Кількість бібліотек та фреймворків для JavaScript постійно збільшується. З'являються гнучкіші рішення, творці яких обіцяють високу продуктивність та швидкодію. Розробникам треба стежити за трендами, щоб через кілька років не виявилось, що вони користуються застарілим стеком і він перестав бути затребуваним. Перелік найбільш відомих фреймворків наведено на рис. 1.2.

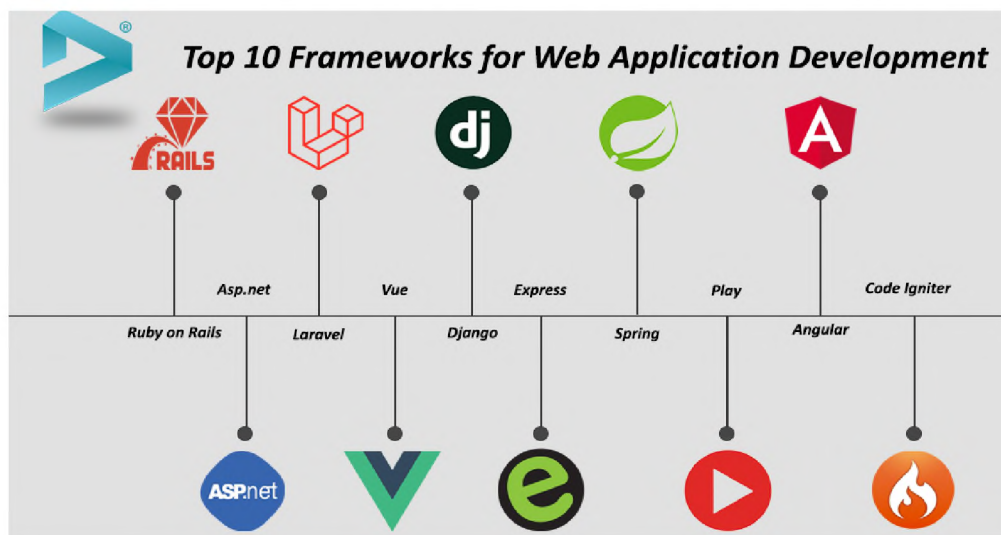


Рисунок 1.2 – Схематичний розподіл відомих фреймворків за призначенням

Відомі аналітичні компанії шляхом різних методів досліджень складають рейтинги популярності фреймворків та бібліотек. Фахівці компанії

Stack Owerflow у 2024 р. провели щорічне опитування серед професійних девелоперів, кількість яких для статистичної вибірки склала 46261 респондента [14]. Результати відповіді на питання: «Над якими веб-фреймворками та вебтехнологіями ви провели значну роботу протягом минулого року?», виявили найпопулярніші вебфреймворки (рис. 1.3).

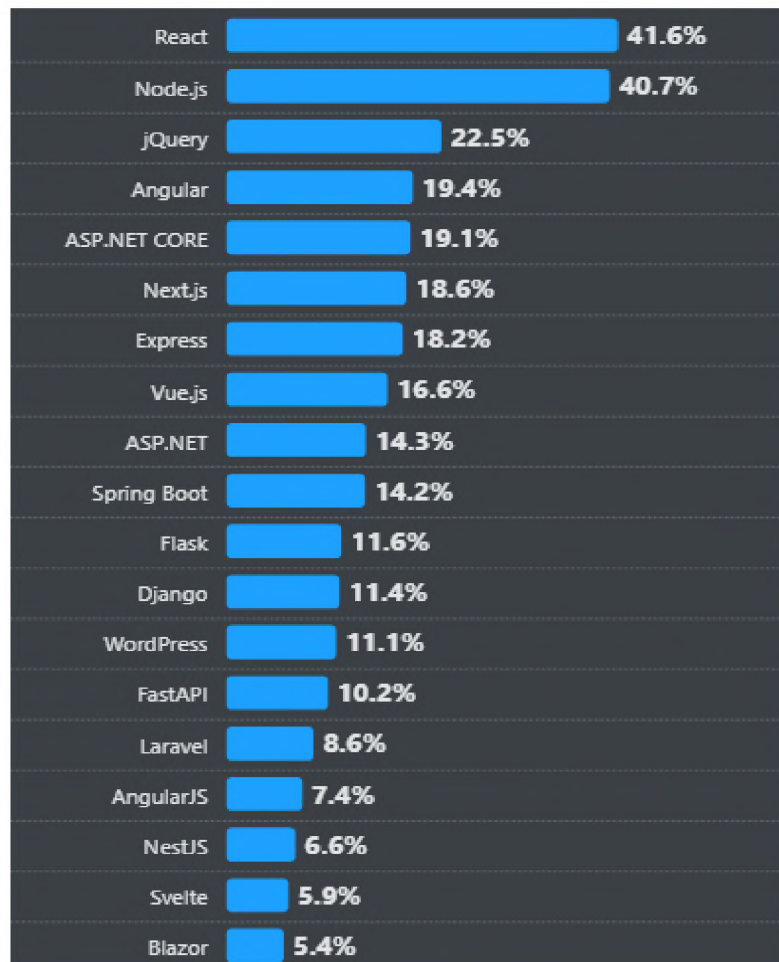


Рисунок 1.3 – Рейтинг відомих фреймворків за популярністю [14]

Опитування включало оцінювання різних технологій за певною шкалою відповідей на питання про різні фреймворки. Потрібно було класифікувати, наприклад, за категоріями: найулюбленіший, найважчий і бажаний; найлегший для навчання та вирішення проблем; найбільш високооплачувані технології та ін. Як видно (див. рис. 1.3), React, Angular не втрачають своїх позицій. Усі вони входять до першої п'ятірки, а React виграв з результатом у 41,6 %. Vue знизив позиції до 16,6 % у порівнянні з 2023 р. [13].

При виборі фреймворка необхідно аналізувати дані з кількох джерел: подивитися статистику в Google Trends, проаналізувати дані щорічного опитування розробників на порталі Stack Overflow, познайомитися з профільними дослідженнями. Графік популярності базований на кількості питань, що стосуються відповідного фреймворку на Stack Overflow станом на травень 2024 р. наведено на рис. 1.4.

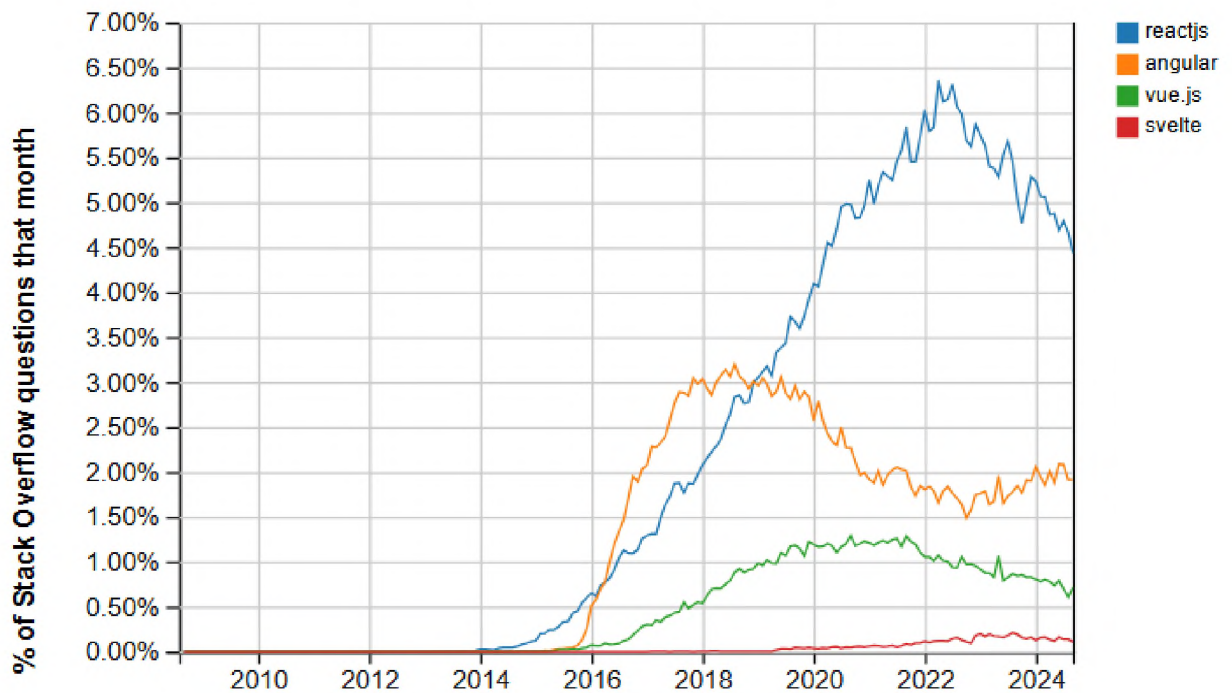


Рисунок 1.4 – Графік популярності фреймворків на Stack Overflow станом на 14.09.2023 [15]

На графіку (див. рисунок 1.3) видно, що найпопулярнішим фреймворком з великим відривом від інших є React, хоча за останній рік він дещо погіршив свої показники. Angular, який був неймовірно популярним в 2016–2018 рр. з тих пір втратив значну частину користувачів, але в нього все ще залишається стабільна база розробників, які надають перевагу саме цьому фреймворку. Vue поступово втрачає свою аудиторію. Новий Svelte ще тільки набирає популярності, але деякі розробники вже вважають його «вбивцею» інших фреймворків.

У фреймворків є переваги і недоліки, тому їх не можна назвати ідеальним інструментом. При правильному використанні вони економлять час, але, якщо у розробника недостатньо досвіду роботи, то у взаємодії з новим фреймворком виникає багато проблем. Завдання, які вирішують фреймворки:

1. Покращують швидкість розробки. У програміста буде відкладене «ядро», яке можна використовувати як основу проекту.

2. Зменшують вартість задач. Якщо програмісту не хочеться створювати сайт з нуля, а можна використовувати фреймворк, то сайт може коштувати дешевше. Самописні CMS можуть розроблятися кілька років, а бюджет часто перевищує розумні значення.

3. Звільняють від рутинних завдань. Розробник може займатися реалізацією нестандартних функцій.

4. Допмагають залишитися конкурентоспроможним на ринку. Якщо програміст в досконалість освоїв кілька популярних фреймворків, він не залишиться без роботи.

У фреймворків відкритий вихідний код, тому будь-який розробник може внести зміни в логіку та адаптувати програмний продукт під свої завдання. Якщо брати готове «ядро», то разом зі стандартними

Для JavaScript на сьогодні розроблено велику кількість бібліотек та фреймворків, їх кількість зростає. З'являються гнучкіші рішення, творці яких обіцяють високу продуктивність та швидкодію. Безумовно, не всі новостворені бібліотеки є абсолютно замінними, не всі мають однакові підходи та можливості. Тому підтвердження рейтинг серед розробників (див. рис. 1.2). Наприклад, серед топових бібліотек і фреймворків для розробки сайтів можна знайти певні відмінності.

React – відкрита JavaScript (JS) бібліотека для розроблення інтерфейсів користувача, яка покликана вирішувати завдання часткового оновлення вмісту вебсторінки. Розробляється командою Meta (раніше Facebook) і доповнюється спільнотою програмістів [16].

Vue.js – JavaScript-фреймворк, що використовує шаблон MVVM для створення інтерфейсів користувача на основі моделей даних, через реактивне зв'язування даних [17].

Angular – фреймворк з відкритим кодом, написаний на TypeScript frontend, який розробляється під керівництвом Angular Team у компанії Google, а також доповнюється спільнотою приватних розробників та корпорацій. Сучасний Angular – це AngularJS, який був переосмислений та перероблений тією ж командою розробників [18].

jQuery – відома JavaScript-бібліотека з відкритим кодом, іншими словами, збірник готових «кодових» рішень. jQuery є однією з найпопулярніших бібліотек для JavaScript, яка посилено використовується на сьогоднішній день і має тисячі прихильників. jQuery є вільним програмним забезпеченням під ліцензією MIT [19].

Backend-частину вебсайту зазвичай пишуть на таких мовах програмування як Java, Python, Ruby, Node.js, PHP. Проте, зараз майже ніхто не використовує мову програмування саму по собі. Переважна більшість backend-розробників в своїй роботі також використовують фреймворки. Найпопулярніші backend-фреймворки станом на 2024 р. наведено на рис. 1.5.

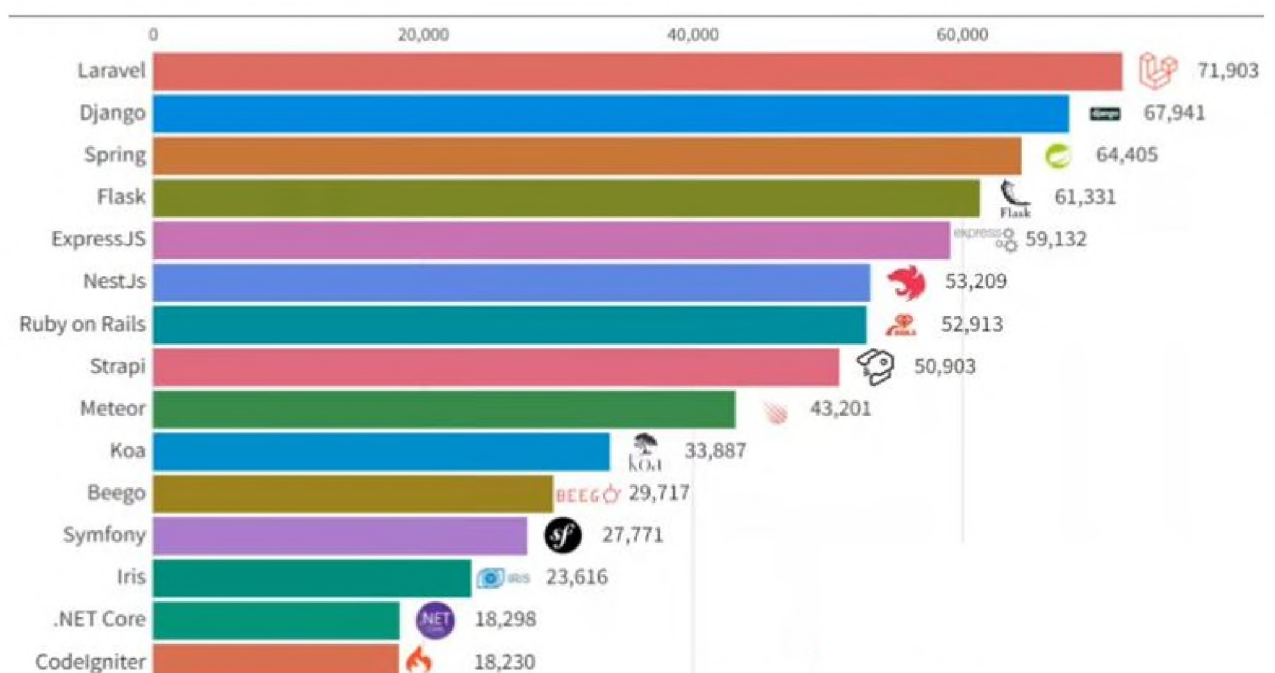


Рисунок 1.5 – Діаграма популярності backend-фреймворків [20]

На діаграмі видно, що найбільш популярними фреймворками є: Laravel (PHP), Django (Python), Spring (Java) та ExpressJS (Node.js).

Вибір є досить суб'єктивним та розрізняється підходами до реалізації при розробці, але всі інструменти для самостійної розробки сайту мають конкурентоспроможні ресурси. Розглянемо детальніше особливості лідера в пошуку – фреймворку React.

### **1.3 Методи раціонального написання оптимального коду при розробці вебдодатку на прикладі використання фреймворку React**

Через постійний розвиток та ускладнення архітектури вебсайтів, вимоги до швидкості, функціоналу та дизайну зростають з кожним роком. Автори публікації [2] зазначають, що невідповідність сучасних тенденціям може серйозно вплинути на взаємодію з користувачем та продуктивність сайту.

Очікується, що вебдодатки будуть доступні цілодобово з будь-якої точки світу та будуть адаптовані для використання з будь-якого пристрою з будь-яким розміром екрану. Окрім цього, важливими складовими сучасного вебдодатку є гнучка і масштабована система безпеки та багатий досвід користувача, побудований на клієнті, вважають автори [21].

Потрібно враховувати, що технології дуже швидко змінюються: те що було популярним вчора, сьогодні вже може бути не актуальним. Втрачають актуальність багатосторінкові сайти, адже швидкість та продуктивність таких сайтів є вкрай низькою. Це пов'язано з тим, що на кожну дію користувача запитується нова сторінка з серверу, та відбувається повне перезавантаження на клієнтській частині. На зміну їм прийшли односторінкові «сайти» Single Page Applications (SPA), які отримують сторінку з сервера лише один раз, а всі інші операції виконуються завдяки асинхронним AJAX та JavaScript скриптам [22]. Подібні зміни вимагали нових ідей та підходів до розробки. Наприклад, поява концепції «компонентів» у веброботці дала змогу створювати

елементи, які можна повторно використовувати в різних частинах коду. Окрім цього, стали з'являтися різні бібліотеки для управління станом додатку (Redux, MobX). Обов'язковою стала можливість інтеграції вебзастосунків з різними платформами і сервісами (соціальні мережі, платіжні системи, API сторонніх сервісів). Для створення real-time застосунків, таких як чати, було створено технологію websockets, яка дозволяє встановлювати постійне двостороннє з'єднання між користувачами, без необхідності постійно створювати нові «коннекти», як при використанні традиційних методів [23].

Саме через необхідність постійного створення нового інноваційного інструментарію для реалізації подібних ідей, сформувалась та невпинно розвивається величезна екосистема вебтехнологій: спеціалізовані мови програмування, фреймворки, бібліотеки, бази даних та системи керування, інтегровані середовища розробки тощо.

Завдяки вебсайту компанія може: допомогти клієнтам в здійсненні оплати, залучати клієнтів до взаємовигідної співпраці, підвищувати свій імідж, аналізувати онлайн-конверсію, визначати цільову аудиторію, формувати статистику та аналітичні звіти, рекламувати свою діяльність, товар або послугу. Розглядаючи важливість вебсайтів у бізнесі, важливо враховувати, що не існує універсального набору інструментів. Різні категорії вебсайтів вимагають різних підходів до створення.

У середовищі розробників помітні суперечки щодо приналежності React до фреймворків або бібліотек. Наприклад, на вебсайті відомого блогера й програміста Еріка Елліота [24] знаходимо визначення (мовою оригіналу): «React is the Top UI Framework in the World». Однак, інформація на офіційному вебсайті чітко говорить про те, що це бібліотека: «JavaScript-бібліотека для створення користувацьких інтерфейсів» [25]. Рекомендації для встановлення програми додаються, тому розпочати роботу в системі можна одразу і безпечно [25]. Документацію про попередні версії системи можна переглянути за спеціальним посиланням. На офіційному сайті викладено необхідні матеріали для вивчення можливостей бібліотеки (рис 1.6).

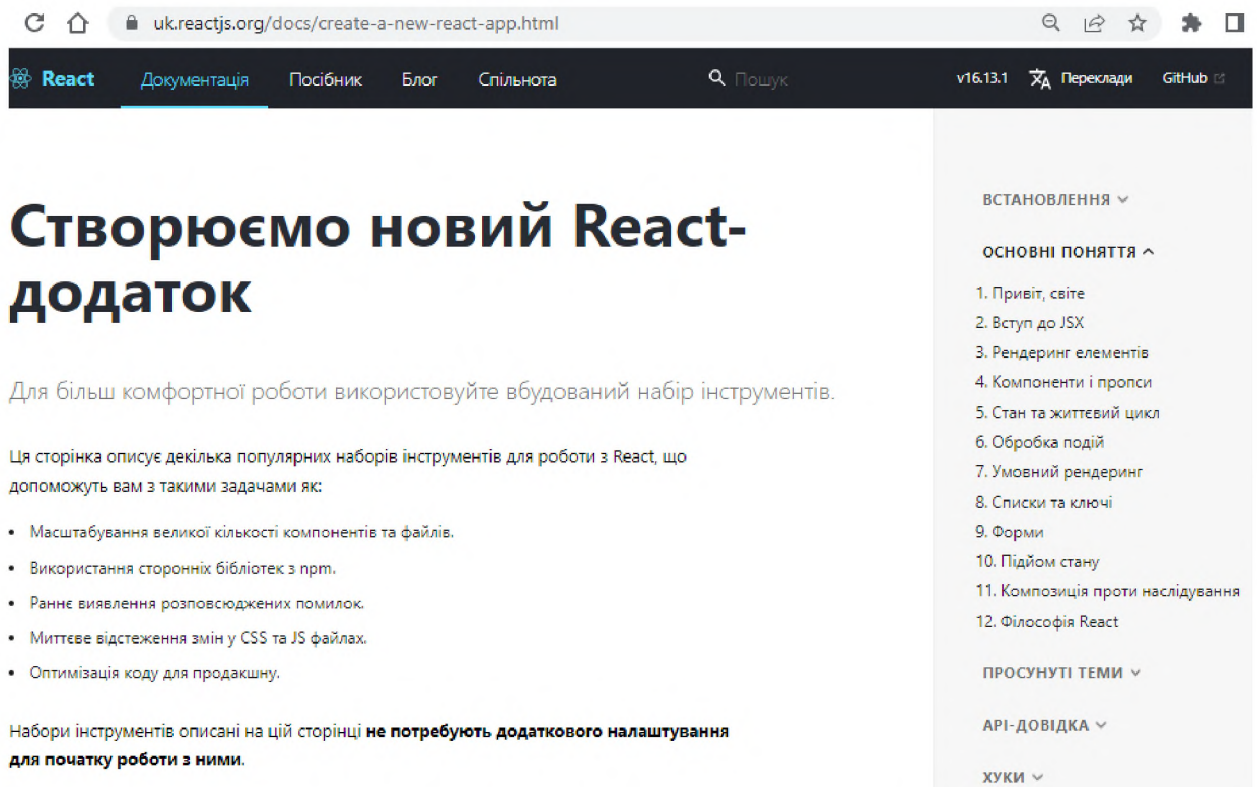


Рисунок 1.6 – Зміст розділу «Документація» на вебсайті React [25]

На офіційному сайті розробників React викладено документацію, навчальні матеріали на багатьох мовах, у т.ч. й українською. Крім того, є тренувальні вправи, а також інформація про попередні версії.

React-додаток будується з компонентів, саме в них і відбувається оновлення і декларація елементів сторінки. Тобто, компонент це головне поняття в React, єдина сутність, яку він містить. Компоненти можуть допомогти розбити інтерфейс на незалежні частини і використовувати їх повторно за потреби та комбінувати будь-яким чином. Це по суті своїй клас, що наслідується з кореня бібліотеки. React-компонент може повертати HTML-код за допомогою JSX. У звичайному JavaScript таке неможливо, це також дуже зручне надлаштування над мовою для зручності розробки.

Компоненти реалізують метод `render()`, який приймає вхідні дані і повертає те, що буде показано користувачу. Приклад застосування методу наведено в коді (рис. 1.7).

```

class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Привіт, {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Віталій" />,
  document.getElementById('hello-example'),
);

```

Рисунок 1.7 – Приклад застосування елементів та методу render()

У цьому прикладі (див. рис. 1.7) використовується XML-подібний синтаксис під назвою JSX. Доступ до вхідних даних, які передаються в компонент, можна отримати за допомогою render() та this.props. Результатом виконання прикладу буде виведення привітання в інтерактивному режимі. Зазвичай, така модель обробки даних поширюється на більш складні мовні інтерактивні взаємодії, і потужність бібліотеки виявляється на повну сутність.

JSX – це препроцесор (розширення JavaScript), що спрощує створення елементів і компонентів React, дозволяє декларативно створювати компоненти користувальницького інтерфейсу. Розширення JSX володіє наступними вагомими можливостями:

- застосування простої декларативною розмітки;
- код розмітки розташований там само, де і код компонента;
- реалізація принципу поділу відповідальностей (наприклад - відділення опису інтерфейсу від логіки стану і від побічних ефектів). При цьому реалізація базується не на використанні різних технологій (наприклад - HTML, CSS, JavaScript);
- абстрагування управління змінами DOM;
- абстрагування від особливостей різних платформ, для яких створюють React-застосунки.

З цим розширенням під час написання коду потрібно набагато менше зусиль, порівняно з класичним JavaScript. JSX трансформується в JavaScript перед запуском у браузері. Він не є обов'язковим під час використання React.

Існує два підходу до розробки – декларативний та імперативний. Суть декларативного підходу в описі кінцевого результату, а імперативного в тому, що ми описуємо конкретні кроки для досягнення результату, тобто яким способом і засобами ми хочемо його досягти. Виявилось, що декларативний підхід добре підходить для розробки інтерфейсів, і саме його притримується React. Завдяки цьому він зміг досягти такої популярності в світі та зайняти провідне місце у веброзробці.

Наприклад, можна порівняти тривіальний приклад лічильника, реалізований імперативним способом за допомогою мови розмітки HTML і JavaScript (рис. 1.8) і декларативним методом за допомогою бібліотеки React (рис. 1.9).

```
<!DOCTYPE html>
<html>

<head>
  <title>Імперативний метод</title>
  <meta charset="UTF-8" />
</head>

<body>
  <div class="root">
    <h1 id="counter-value"></h1>
    <button id="increment-btn">+1</button>
  </div>

  <script>
    const counterValueElement = document.getElementById("counter-value");
    const incrementBtn = document.getElementById("increment-btn");
    let counterValue = 0;

    function increment() {
      counterValue += 1;
      counterValueElement.innerText = counterValue;
    }

    counterValueElement.innerText = counterValue;
    incrementBtn.addEventListener("click", increment);
  </script>
</body>
</html>
```

Рисунок 1.8 – Приклад декларативного підходу до веброзмітки

```

<!DOCTYPE html>
<html>

<head>
  <title>Декларативний метод</title>
  <meta charset="UTF-8" />
</head>

<body>
  <div id="root"></div>
  <script src="https://unpkg.com/react@17/umd/react.development.js" crossorigin></script>
  <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js" crossorigin></script>
  <script src="https://unpkg.com/@babel/standalone@7.13.6/babel.min.js" crossorigin></script>
  <script type="text/babel">
    function App() { const [value, setValue] = React.useState(0); function increment() { setValue
    (value + 1); } return (
    <div className="app">
      <h1>{value}</h1>
      <button onClick={increment}>+1</button>
    </div>
    ); } ReactDOM.render(
    <div>
      <App />
    </div>, document.getElementById("root") );
  </script>
</body>
</html>

```

Рисунок 1.9 – Приклад імперативного підходу з використанням

Як бачимо, імперативний метод є більш компактним і візуально зрозумілим, але має необхідність зберігати елементи на сторінці в змінні, яких зазвичай набагато більше, а от в React рендер відбувається в самому компоненті без необхідності декларування змінних і їх збереження.

Отже, можна зробити висновок, що фреймворк React є надзвичайно зручним інструментом у веброзробці і за допомогою неї можна будувати сучасні та швидкі вебдодатки.

Файли CSS можна завантажити в заголовок вашої сторінки для загальних глобальних макетів, шрифтів тощо. Вони чудово працюють.

Модулі CSS – це локальні файли CSS, які можна імпортувати безпосередньо у файли JavaScript. Для цього знадобиться правильно налаштований завантажувач. Next.js увімкне це за замовчуванням.

Styled-jsx дозволяє оголошувати вбудовані стилі у компонентах React, подібно до того, як теги <style> працюють у HTML. Сфера застосування цих стилів є гіперлокальною, що означає, що стилі впливатимуть лише на рідні теги та їхні дочірні теги. Next.js також включає styled-jsx за замовчуванням.

Virtual DOM у React представляє легковажну копію звичайного DOM. І відмінною особливістю React є те, що ця бібліотека працює саме з віртуальним DOM, а не звичайним.

Якщо програмі необхідно уточнити інформацію про стан елементів, то вона звертається до віртуального DOM. За рахунок Virtual DOM має збільшуватися продуктивність.

React надає декларативний API і це створює умови, за яких не потрібно турбуватися про зміни, які відбуваються, що спрощує написання додатків.

React надає повний контроль над розміром додатка, дозволяючи включити тільки ті речі, які дійсно потрібні. Він дає більше гнучкості під час переходу від односторонніх додатків (SPA) до мікросервісів, використовуючи частини колишнього додатка.

React чудово підійде для проєктів, де важлива гнучкість при створенні великих екосистем, що мають тенденцію розростатися. Дуже добре підходить для командної розробки. UI-код читабельний і простий у супроводі. Здатний повністю взяти на себе відповідальність за рівень подання в архітектурі MVC (модель, подання, контролер). Розробка користувацького інтерфейсу за цим принципом – це сучасний підхід у сфері надання даних.

Окремо важливим моментом є адаптивність. Зараз популярним є підхід «Mobile First». Автори роботи [26] закликають завжди дотримуватися цього принципу. Цей принцип означає, що застосунки повинні розроблятися таким чином, щоб в першу чергу гарно виглядати на мобільних пристроях, а вже після цього на широких екранах.

Сайт, створений за принципом Mobile First має наступні переваги: високе ранжування, швидке завантаження, сприяння здійсненню швидкої покупки. До появи таких інструментів як CSS та JavaScript такий підхід, та й взагалі адаптивність вебсторінки була неможливою. Це важливо, оскільки якщо подивитись на статистику інтернет-трафіку за 2023 рік, то можна побачити, що трохи більше ніж 60% трафіку складають користувачі мобільних пристроїв.

Крім того, розвиток мобільних технологій також призводить до необхідності впровадження в розробку специфічних фреймворків та інструментів, таких як React Native, Flutter або Xamarin. За допомогою цих інструментів розробники можуть створювати кроссплатформені додатки, які працюють як в браузерах, так і на iOS та Android. Такий підхід дозволяє писати один код, який буде виконуватись на різних видах девайсів, що суттєво економить фінансові ресурси замовника та час розробника [27].

Враховуючи стрімкий розвиток галузі та постійну появу нових інноваційних інструментів, а також велику конкуренцію між вебстудіями, одним із головних завдань сучасного веброзробника є аналіз та підбір найкращого рішення саме для потреб його проєкту.

### **Висновки до розділу 1**

В теретичному розділі було проаналізовано сучасний стан вебтехнологій та тенденції їх розвитку. На основі даних світових аналітичних компаній можемо стверджувати, що наразі відбувається стрімкий розвиток вимог та інструментів для розробки вебдодатків, а також, що вебсайти є невід'ємною частиною існування сучасного бізнесу.

Було розглянуто особливості використання різних інструментів сучасного веброзробника. Також були представлені різні графіки популярності та порівняльні таблиці відповідних технологій, на які можна опиратися під час вибору інструментарію.

React це одна з найбільш популярних у світі JavaScript-бібліотек, яка призначена для генерації динамічних інтерфейсів користувача та користувацьких інтерфейсів вебдодатків. Поява цієї бібліотеки на фундаментальному рівні змінило спосіб роботи JavaScript-фреймворків і бібліотек. Завдяки перевикористанню коду стало набагато простіше створювати мобільні застосунки.

## РОЗДІЛ 2

# ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ JAVASCRIPT ФРЕЙМВОРКІВ ТА МОВНИХ МОДЕЛЕЙ ШТУЧНОГО ІНТЕЛЕКТУ ЩОДО ОПТИМІЗАЦІЇ ПРОГРАМНОГО КОДУ ВЕБДОДАТКІВ

### 2.1 Дослідження продуктивності популярних фреймворків JavaScript та можливостей досягнення оптимального результату

Для досягнення мети кваліфікаційної роботи було сформульовано низку завдань, основною ідеєю яких є дослідження кількох концептуально різних засобів досягнення оптимального результату при створенні програмного коду у веброзробці, таких як різні фреймворки мови JavaScript, а також можливості мовних моделей штучного інтелекту, навчених на різних бібліотеках мов програмування. На першому етапі проведено тестування для визначення основних переваг та особливостей використання фреймворків мови JavaScript (JS фреймворків), а також проведення аналізу їх швидкодії.

Задача правильного вибору фреймворку виникає на кожному етапі розробки, починаючи від розробки ідеї та архітектури програмного продукту і закінчуючи фазою підтримки та післярелізного обслуговування. Відповідно, правильний вибір впливає на функціональність, продуктивність, безпеку, підтримку та життєвий цикл програмного продукту. В сучасному програмному інженерінгу та інформаційних технологіях дослідження критеріїв вибору фреймворків є актуальним, оскільки сприяє ефективному використанню ресурсів та досягненню високої якості програмного забезпечення. Для вибору методів дослідження вивчалися відомі результати, що були опубліковані раніше в науковій літературі.

У роботі [22] було проведено порівняльний аналіз продуктивності інструментів React та Vue.js, які часто використовують для створення односторінкових додатків, або Single Page Application (SPA). Дослідження проводилося за допомогою двох додатків з однаковою функціональністю, реалізованих на React і Vue.js. Автори вимірювали продуктивність SPA за допомогою інструментів, доступних у вебпереглядачах, і відповідних

реалізаціях їх власних методів. Згідно результатів дослідження React трохи перевершує Vue.js для компонентів з малою кількістю елементів. Однак, це змінюється, якщо сторінка, яка створюється, має більшу кількість елементів. Тоді Vue.js ефективніший. Відмінності між браузерами очевидні. Додатки, створені для дослідження, добре показують, що кількість даних, представлених на сторінці, а також браузер, який використовує кінцевий користувач, мають значний вплив на продуктивність. Згідно з результатами вимірювань тестових додатків, у випадку з Chrome механізм виконання JavaScript є більш ефективним, а його код більш оптимальним порівняно з аналогом у Firefox.

Результат порівняння двох сучасних Python фреймворків – Django та Flask за критеріями часу написання програмних модулів та об'ємом програмного коду представлений в публікації [23]. Аналіз показав, що фреймворк Django є кращим за об'ємом програмного коду, механізмом обробки формулярів, часом розробки додатків. Від вибору правильного веб-фреймворка залежить якість, надійність та час реалізації кожного розроблюваного програмного засобу. Високий попит на вебдодатки змушує розробників писати економічно ефективні, безпечні та оптимізовані вебдодатки.

Порівняння найпопулярніших PHP фреймворків, яке представлено в статті [28], було здійснене на основі методу QSOS (Qualification and Selection of Opensource Software), який офіційно визнається стандартом [29]. Результати показали, що фреймворки Laravel та Symfony задовільняють сучасним стандартам та вимогам, тоді як CodeInginter їм не відповідає. Порівняння продуктивності кожного з фреймворків здійснювалося за такими критеріями, як кількість запитів за секунду, використання пам'яті, час відгуку та кількість необхідних файлів. Для цього авторами було використано спеціальний дизайн вебсторінки та Apache Benchmark. Дослідження показало, що Laravel перевершує інші MVC фреймворки. Для початківців більше підходить Laravel, ніж Symfony, який призначений для досвідчених розробників і розробки

комплексних вебзастосунків. CodeIgniter найкраще підійде при розробці простих і коротких проєктів.

Автори роботи [30] дослідили еволюцію популярності та провели багатоаспектне порівняння десяти найбільш широко використовуваних веб-фреймворків за останні п'ятнадцять років. Базою дослідження виступала кількість новостворених репозиторіїв на GitHub, а також кількість запитів на Stack Overflow, які стосувалися відповідних фреймворків. Залежність між цими двома факторами проаналізована в контексті популярності мов програмування на основі індексу TIOBE. Тест Грейнджера підтвердив кореляцію між двома джерелами, за винятком ASP.NET та React.js при отриманні даних з GitHub, а також за винятком React.js при зборі даних із Stack Overflow.

Автори [31] проаналізували складності використання та продуктивності вебзастосунків, побудованих на основі Java фреймворків: Spring Boot, Micronaut, Quarkus та Javalin. Фреймворки Spring Boot, Micronaut та Quarkus представляють схожі конвенції та підходи до стилю розробки програм. Серед цих трьох фреймворків виділяється Quarkus, який має багато корисних функцій, таких як «гаряче розгортання» або спеціальний режим розробки. Результати дослідження явно вказують на те, що додаток на основі фреймворку Javalin є найефективнішим порівняно з іншими.

Фреймворк Spring Boot отримав найгірші результати тестування, проте він широко використовується завдяки високій функціональності та набору бібліотек і готових рішень, підтримуваних стандартів.

Для проведення дослідження продуктивності популярних фреймворків Angular, React, Svelte та Vue було проведено два різних тести – під високим та низьким навантаженням. В першому тесті був використаний JS-framework-benchmark [32]. Мета цього скрипту – перевірка швидкодії frontend фреймворків. Заміри результатів відбуваються за допомогою вбудованих функцій.

Порівняння проводилось за наступними показниками:

- create rows – час необхідний для створення 1000 елементів після первинного завантаження сторінки (без підготовки);
- replace all rows – час необхідний для заміни всіх 1000 елементів в таблиці (з 5 підготовчими ітераціями);
- partial update – час необхідний для оновлення тексту в кожному десятому рядку в таблиці з 10000 елементами (з 5 підготовчими ітераціями);
- select row – час необхідний для виділення рядку у відповідь на клацання по рядку (з 5 підготовчими ітераціями);
- swap rows – час необхідний для перестановки 2 рядків в таблиці з 1000 елементами (з 5 підготовчими ітераціями);
- remove row – час необхідний для видалення рядку з таблиці з 1000 елементами (з 5 підготовчими ітераціями);
- create many rows – час необхідний для створення 10000 елементів (без підготовки);
- clear rows - час необхідний для очищення таблиці з 10000 елементів (без підготовки).

По кожному з показників було пройдено по 10 ітерацій на кожен тест, після чого було пораховано середнє значення кожного з них. Тестування проводилось в автоматичному режимі, що виключало фактор людського впливу. Для чистоти експерименту перед кожною ітерацією тестів робоча станція перезавантажувалась. Окрім цього, був відключений доступ в Інтернет, та не допускалась фонові робота будь-якого стороннього програмного забезпечення.

Тестування проводилось локально, в режимі браузеру «інкогніто». Була використана остання на момент проведення тестування (15.01.2024) версія Google Chrome, а також сама актуальна версія Windows 10 Home Edition. Детальні характеристики комп'ютера наведено в таблиці 1.

Тестування проводилось з використанням наступних версій фреймворків: Angular-v15.0.1, React-v18.2.0, Svelte-v4.0.0, Vue-v3.3.4.

Таблиця 2.1 – Характеристики робочої станції

Елемент характеристики	Тип, версія забезпечення
Центральний процесор	AMD Ryzen 5 4600H, 3.0GHz (6 cores, 12 threads)
Обсяг оперативної пам'яті	16GB, DDR4-3200 MHz
Тип накопичувача	SSD
Операційна система	Windows 10 Home
Веббраузер	Google Chrome 117.0.5938.89 (Official build) (64-bit)

Результати тестування представлені в табл. 2.2. Всі числові значення наведені в мілісекундах, тобто: 123.2 = 0.1232 секунд. Зеленим кольором виділено найкращий результат в кожному тестуванні, червоним – найгірший.

Таблиця 2.2 - Результати тестування JS-framework-benchmark

Тестові операції	Типи JS-фреймворків, які підлягали тестуванню			
	Angular-v15.0.1	React-v18.2.0	Svelte-v4.0.0	Vue-v3.3.4
create rows, мс	123.2	123.3	117.9	113.8
replace all rows, мс	133.3	128.9	123.2	118.9
partial update, мс	541.0	596.3	545.1	612.4
select row, мс	89.7	114.8	100.6	98.3
swap rows, мс	520.7	452.7	80.5	81.8
remove row, мс	135.5	142.5	134.4	144.3
create many rows, мс	1246.5	1378.3	1223.9	1152.6
clear rows, мс	165.7	94.2	111.5	100.6
Разом, мс	2955.6	3031.0	2437.1	2422.7

Для більш наочного відображення, всі дані було експортовано в .xlsx та представлено у вигляді діаграми (рис. 2.1).

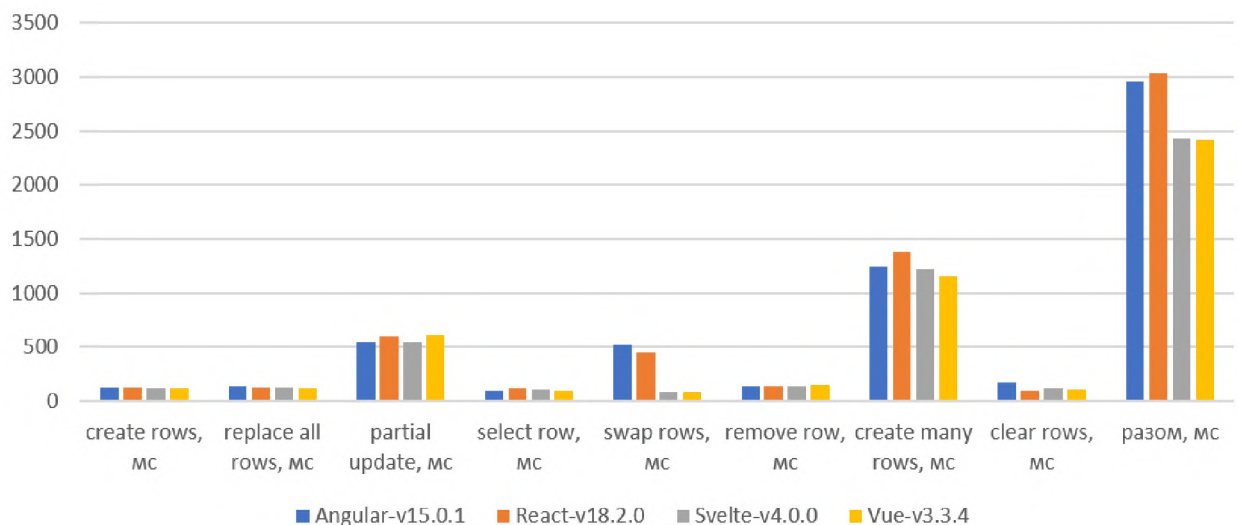


Рисунок 2.1 – Результати тестування за допомогою JS-framework-benchmark

Результати вимірювань виявились дещо різними. Angular та React показали найгірші результати майже в усіх тестах, за винятком деяких: React краще всіх впорався з очисткою рядків, а Angular проявив себе при виборі рядків та частковому оновленні тексту. Причиною цього може бути повільна робота з Virtual DOM. Найкращі показники у Vue та Svelte, Vue був швидше в більшій кількості тестів, але Svelte не був найгіршим в жодному з них. Загалом найкращим виявився Vue, а найгіршим React.

Для другого тесту було створено чотири простих, функціонально ідентичних тестових додатки, реалізованих на кожному з розглянутих раніше фреймворків. Зберігання інформації в цих додатках відбувається в Local Storage, що виключає вплив на результати часу передачі інформації з сервера до клієнта. Заміри результатів в цьому тесті здійснювались за допомогою Chrome DevTools [33]. Програмний код для тестування на кожному з фреймворків реалізований максимально просто, дотримуючись рекомендацій, наведених у відповідній офіційній документації. Це потрібно, щоб мінімізувати людський вплив на результати тестування. Дослідження проводилось з використанням тієї самої робочої станції та програмного забезпечення, що і в попередньому тестуванні. Під час дослідження брались до уваги наступні показники Chrome DevTools:

- System: системні операції;
- Scripting: виконання JavaScript коду;
- Rendering: формування сторінки в оперативній пам'яті;
- Painting: відображення сторінки.

Функціонал створених тестових додатків включає наступні функції:

- створення та відображення 1000 випадкових текстових рядків. Під час виконання цієї дії у фреймворку немає можливості використовувати попередньо створені елементи, тому ця операція є ресурсовитратною;

- видалення всіх рядків. Ця операція прибирає з екрану та пам'яті застосунку всі раніше створені елементи;
- зміна одного випадкового рядка на інший.

Під час тестування кожна функція була виконана по 20 разів. У табл. 2.3 показана сумарна тривалість операцій.

Таблиця 2.3 – Результати тестування за допомогою Chrome DevTools

Показники для тестування	Типи JS-фреймворків, які підлягали тестуванню			
	Angular-v15.0.1	React-v18.2.0	Svelte-v4.0.0	Vue-v3.3.4
System, мс	940	680	1090	1250
Scripting, мс	620	1160	700	480
Rendering, мс	970	870	810	930
Painting, мс	300	200	150	250
Разом, мс	2830	2910	2750	2910

Для більш наглядного відображення, всі дані було експортовано в .xlsx та представлено у вигляді діаграми (рис. 2.2).

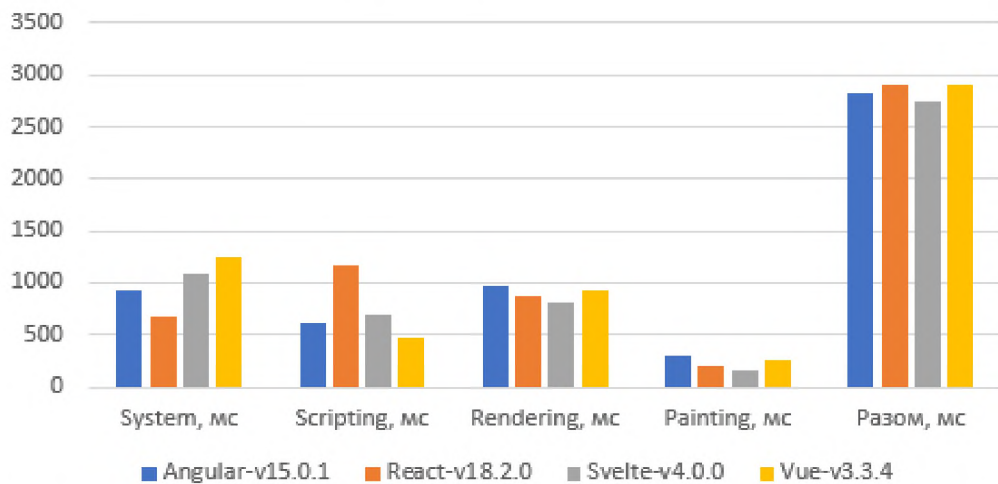


Рисунок 2.2 - Результати тестування за допомогою Chrome DevTools

У другому тесті всі фреймворки показали майже однакові сумарні результати, але різні за окремими критеріями. Зокрема React слабо впорався зі Scripting, але перевершив всіх в System, що свідчить про суттєві відмінності в архітектурі фреймворків. Причиною хорошого результату Svelte є те, що він не використовує Virtual DOM на відміну від інших, а відразу компілюється в низькорівневий код. Таким чином, дослідження показують, що використання різних фреймворків дає очікувано високий результат при написанні коду, якщо досвід програміста дозволяє скласти своєрідний «пазл», в якому співпадають поставлені задачі та обрані технології.

## 2.2 Дослідження можливостей використання штучного інтелекту в області програмної інженерії

За останні пару років штучний інтелект (ШІ) став свідком експоненціального зростання в розвитку, збільшення використання ШІ та підвищення суспільного інтересу на різних рівнях, від індивідуального до організаційного [34-36]. Сучасний штучний інтелект використовує машинне навчання та інші передові методи для створення нових знань, контенту, гіпотез і навіть інноваційних ідей шляхом визначення шаблонів та інформації, яка зазвичай міститься у великих базах даних. Це стало каталізатором використання штучного інтелекту в широкому спектрі програм, включаючи проектування та розробку програмного забезпечення (ПЗ) [37]. Відтак, говорячи про шляхи найбільш раціонального написання коду, не можна не висвітлити застосування штучного інтелекту. Останнім часом з'являються публікації, в яких наведено приклади використання штучного інтелекту на основі великих мовних моделей (Large Language Models, LLM) для виконання різноманітних завдань: від навчання на великих даних з подальшою обробкою в прикладних інформаційних системах [38], аналізу функціоналу прикладних систем на основі сформульованих вимог до [39], до виправлення помилок і генерації нового коду із заданими функціями.

Автори ландшафтного дослідження кількарічних публікацій [40], пов'язаних із використанням ШІ в різних аспектах програмної інженерії, відзначають, що науковці почали широко впроваджувати нові підходи на основі штучного інтелекту для вирішення традиційних проблем програмної інженерії, зокрема, на основі мовних моделей [41-42]. Джерелом аналізу стали повторювані тематичні ключові слова авторів у публікаціях, контент, категорії, теми. В ролі вихідної інформаційної бібліографічної бази було взято Scopus (Elsevier, Амстердам, Нідерланди), оскільки вона вважається найбільшою базою рефератів і цитат рецензованої наукової літератури. На додаток до розширених аналітичних служб, він дозволяє експортувати 20 000

записів одночасно. Пошуковий запит включав всі можливі комбінації ключових слів, які можуть бути пов'язані з розробкою програмних кодів та штучним інтелектом, а саме:

TITLE-ABS-KEY((«штучний інтелект» АБО «машинне навчання» АБО «глибоке навчання» АБО «інтелектуальна система» АБО «машина опорних векторів» АБО («дерево рішень» І («індукційний» АБО «евристичний»)) АБО «Марковський процес прийняття рішень» АБО «прихована модель Маркова» АБО «нечітка логіка» АБО «k-найближчий сусід» АБО «наївний Байєс» АБО «Байєсівське навчання» АБО «штучна нейронна мережа» АБО «згорточна нейронна мережа» АБО «рекурентна нейронна мережа» АБО «генеративна змагальна мережа» АБО «мережа глибоких вірувань» АБО «персептрон» АБО {обробка природної мови} АБО {розуміння природної мови} АБО {загальна модель мови}) і ({розробка програмного забезпечення} АБО {дизайн програмного забезпечення} або { розробка програмного забезпечення})) І PUBYEAR > 2018 І PUBYEAR < 2025.

Розподіл за типами статей показує, що більшість публікацій були доповідями, пов'язаними з конференціями, що свідчить про те, що дослідження все ще перебуває на стадії дозрівання та що основні знання ще формуються. Цей висновок також підтверджується тим фактом, що трьома найбільш плідними назвами є матеріали конференцій, що вказує на те, що список основних журналів ще потрібно створити.

Тенденція продуктивності досліджень, яка представлена на рис. 2.3, свідчить, що пік продуктивності загальної кількості публікацій був досягнутий у 2020 р. Однак кількість публікацій стабілізувалася у 2022 р. Зменшення кількості публікацій відповідає загальній тенденції продуктивності програмного забезпечення інженерних публікацій, індексованих у Scopus, які також почали демонструвати спад у 2020 р., головним чином через зменшення кількості доповідей на конференціях (сплеск пандемії), тоді як кількість статей почала збільшуватися у 2022 р. Наведені вище дані можуть свідчити про початок позитивної тенденції до досягнення дослідницької зрілості.

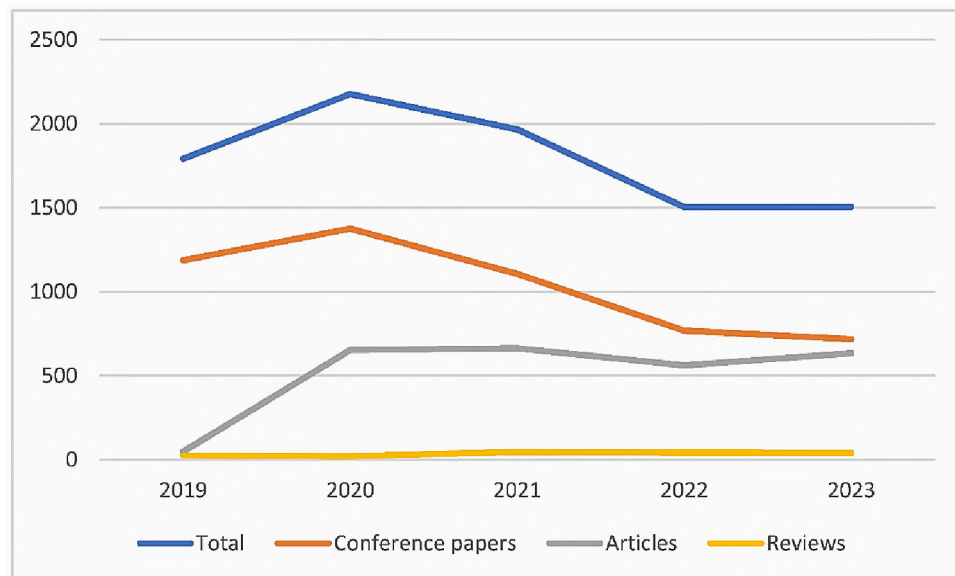


Рисунок 2.3 – Динаміка продуктивності наукової літератури в області ІІІ [40]

Щодо географії досліджень, то, якщо позначити  $n$  як кількість публікацій, пов'язаних з ІІІ, то найпродуктивнішими країнами був Китай ( $n = 2042$ ), за ним йшли Сполучені Штати ( $n = 1193$ ), Індія ( $n = 934$ ), Німеччина ( $n = 445$ ) і Канада ( $n = 381$ ). Це відповідає рейтингу Scimago Country Rankings (Elsevier, Амстердам, Нідерланди), де Сполучені Штати посідають перше місце за програмним забезпеченням і друге за штучним інтелектом, Китай посідає перше місце за штучним інтелектом і друге за програмним забезпеченням, а інші провідні країни займають серед 10 найрезультативніших в обох категоріях. Усі найбільш продуктивні країни також входять до G20 [43].

Аналіз вмісту найбільш плідних дослідницьких тем проводили за допомогою SKS Steps 3 і 4 та програмного забезпечення VOSViewer, версія 1.6.20 (Лейденський університет, Лейден, Нідерланди). Ландшафт авторських ключових слів показаний в додатку А, а синтез результатів наведено в таблиці Б.1 додатку Б. Контент-аналіз ландшафту дослідження складається з 146 ключових слів автора (див. додаток А), що призводить до 15 категорій і 5 тем.

Найбільш популярними за кількістю публікацій виявилися словосполучення і терміни, а також пов'язані з ними області досліджень, які наведені в табл. 2.4.

Таблиця 2.4 – Кластери та пов'язані з ними ключові слова і категорії

Репрезентативні ключові слова, к-ть	Категорії (області дослідження)
Штучний інтелект (560), розробка програмного забезпечення (173), тестування програмного забезпечення (123), нечітка логіка (98), програмне забезпечення (73), великі дані (65), навчання з підкріпленням (64)	Етичне використання розробки програмного забезпечення на основі штучного інтелекту, використання нечіткої логіки в розробці та тестуванні програмного забезпечення, автоматизація тестування програмного забезпечення в гнучкому середовищі, управління проектом життєвого циклу програмного забезпечення за допомогою нечіткої логіки, наука про дані та великі дані в розробці програмного забезпечення
Розробка програмного забезпечення (673), обробка природної мови (362), розробка вимог (108), гнучка розробка програмного забезпечення (61)	Обробка природної мови при розробці програмного забезпечення, обробка природної мови при розробці вимог до програмного забезпечення, розуміння історій користувачів за допомогою обробки природної мови
Машинне навчання (1504), оцінка зусиль розробки програмного забезпечення (156), класифікація (142), передбачення дефектів програмного забезпечення (205), аналіз даних (102), штучна нейронна мережа (184), показники програмного забезпечення (84), вибір функцій (82)	Оцінка зусиль щодо розробки програмного забезпечення, інтелектуальний аналіз даних у прогнозуванні помилок/дефектів програмного забезпечення, машинне навчання та метрика програмного забезпечення
Якість програмного забезпечення (86), обслуговування програмного забезпечення (62), сховища програмного забезпечення для майнінгу (43)	Репозиторії програмного забезпечення для майнінгу для покращення якості програмного забезпечення та обслуговування програмного забезпечення, Crowdsourcing, GitHub та програмне забезпечення з відкритим вихідним кодом як джерела для даних розробки програмного забезпечення для майнінгу

Як видно з табл. 2.4, найбільш активно за останні 5 років розвивалися напрямки як розробки самих штучних нейронних мереж, машинного навчання, так і використання систем ШІ в областях обробки природної мови при розробці програмного забезпечення, при обробці великих даних. Показово, що на порядки менше було застосовано тих самих тем в областях обслуговування репозитаріїв коду, використання ПЗ з відкритим кодом для розробки ПЗ, прогнозування помилок, підвищення якості програмного забезпечення.

На думку багатьох дослідників, обробка природної мови (Natural language processing, NLP) в розробці ПЗ може значно покращити завдання

розробки програмного забезпечення [44]. NLP може підтримувати категоризацію помилок [45], розробку більш безпечного програмного забезпечення, декомпозицію програми [46], класифікацію зобов'язань [47], програмування та кодування [48], написання узгоджених і фактично правильних читань [49], розробку на основі моделі [50], розгортання шаблонів проектування [51] та управління відстежуваністю [52].

Загалом, вивчення попередніх досліджень показали, що дослідження штучного інтелекту значно вплинули на розробку програмного забезпечення за останні роки. Від обробки природної мови в розробці ПЗ, використання ШІ в управлінні життєвим циклом розробки ПЗ, використання машинного навчання для прогнозування помилок/дефектів і оцінки зусиль, використання глибокого навчання в інтелектуальній розробці ПЗ та управлінні кодом, до репозиторіїв ПЗ для майнінгу для покращення якості програмного забезпечення, ШІ змінив спосіб створення ПЗ розробниками та інженерами. Конвергенція ШІ та розробки ПЗ має потенціал для значного скорочення необхідних ресурсів, покращення якості та покращення взаємодії з користувачем за допомогою більш інтелектуальних програм, орієнтованих на користувача

У розглянутих роботах відзначено, що на сьогодні все ще потрібне загальне розуміння поточного стану, можливих цільових застосувань, практичних сценаріїв використання штучного інтелекту в програмній інженерії, неминучих обмежень, та інших проблем.

### **2.3 Характеристики сучасних моделей ШІ**

Найбільш популярною на сьогодні в різних областях застосування є модель ШІ GPT-4, яка стала наступницею попередньої версії GPT-3.5. Розробкою займається компанія OpenAI [53]. Можливості моделі варто розглянути через порівняння версій. GPT-4 – це велика мультимодальна модель (приймає вхідні зображення та текст, видає текстові виходи), яка, хоча

й менш здатна, ніж люди в багатьох сценаріях реального світу, демонструє продуктивність на рівні людини на різних професійних і академічних тестах. Наприклад, він здає симуляцію адвокатського іспиту, набравши приблизно 10% найкращих учасників іспиту; навпаки, оцінка GPT-3.5 була близько нижніх 10%. Розробники витратили 6 місяців на ітераційне узгодження GPT-4, використовуючи уроки змагальної програми тестування, а також ChatGPT, що призвело до найкращих результатів (хоча й далеко не ідеальних) щодо фактичності, керованості та відмови виходити за поручні.

Протягом останніх двох років перебудували весь стек глибокого навчання та разом із Azure спільно з нуля розробили суперкомп'ютер для підтримки робочого навантаження. У результаті тренувальний запуск GPT-4 був за оцінкою розробників безпрецедентно стабільним, ставши першою великою моделлю, продуктивність навчання якої вдалося точно передбачити заздалегідь.

Як і попередні моделі GPT, базову модель GPT-4 було навчено передбачати наступне слово в документі з використанням загальнодоступних даних (наприклад, даних Інтернету), а також даних, які ми отримали ліцензію. Дані — це сукупність даних у веб-масштабі, включаючи правильні та неправильні розв'язки математичних задач, слабкі та сильні міркування, суперечливі та послідовні твердження, які представляють велику різноманітність ідеологій та ідей.

Тож на запит із запитанням базова модель може відповідати різними способами, які можуть бути далекими від намірів користувача. Щоб узгодити її з наміром користувача в межах огорожень, ми точно налаштуємо поведінку моделі за допомогою навчання з підкріпленням із зворотним зв'язком людини (RLHF).

Щоб зрозуміти різницю між цими двома моделями, були проведені дослідження за допомогою різноманітних тестів, у тому числі симуляції іспитів, які спочатку були розроблені для людей. Були використані найновіші загальнодоступні тести (у випадку олімпіад і питань з безкоштовними

відповідями або придбавши видання практичних іспитів 2022–2023 рр. Результати наведені у вигляді графіку в додатку В.

В табл. 2.5 представлено результати оцінки GPT-4 (за [53]) з найбільш відомими моделями (state-of-the-art, SOTA) та мовними моделями (LM SOTA) за традиційними тестами, розробленими для моделей машинного навчання.

Таблиця 2.5 – Порівняльні результати тестування моделі GPT-4

Тип тесту	GPT-4	GPT-3.5	LM SOTA	SOTA
Завдання з вибором відповідей із 57 предметів (професійні та академічні)	86,4%	70,0%	70,7%	75,2%
Розумні міркування навколо повсякденних подій	95,3%	85,5%	84,2%	85,6%
Питання з природничих наук для початкової школи. Виклик-набір	96,3%	85,2%	82,4%	85,6%
Розсудливі міркування щодо розділення займенників	87,5%	81,6%	84,2%	85,6%
Завдання з програмування на мові Python	67,0%	48,1%	26,2%	65,8%
Розуміння прочитаного та арифметика	80,9%	64,1%	70,8%	88,4%

Як видно з табл. 2.5, модель GPT-4 показала найкращі результати у всіх типах тесту порівняно з іншими, у т.ч. мовними, моделями. Найнижчий результат отримано при вирішенні завдання на програмування мовою Python, однак, він також найкращий серед інших моделей. Можна припустити, що можливості моделі впливають здебільшого з процесу попереднього навчання. Але керування моделлю відбувається після процесу навчання – базова модель потребує швидкого проектування, щоб навіть знати, що вона має відповідати на запитання.

Щодо створення програмного коду, то віднедавна мовні моделі сприяють прогресу вирішення давньої проблеми програмного синтезу [54], що викликано наявністю коду у великих наборах даних. На основі успіху мовних моделей в інших видах модальності, автори роботи [55] припустили, що спеціалізована модель GPT, яка називається Codex, може досягти успіху в різноманітному кодуванні завдань. Показано метод навчання моделі

кодувальника-декодера на основі мільйонів виправлень помилок в історії змін проєктів, розміщених на GitHub.

Увага була зосереджена на задачі генерації автономних Python-функцій з docstring та оцінюванні правильності зразків коду автоматично через юніт-тести. Це контрастує з генерацією природної мови, де зразки зазвичай оцінюються за допомогою евристик або людських оцінювачів. Для точного бенчмаркінгу обраної моделі створювався набір даних з 164 оригінальних програмних задач з юніт-тестами. Ці задачі оцінюють розуміння мови, алгоритми та просту математику, з деякими, що є порівнянними з простими питаннями на співбесідах для програмістів. Щоб розв'язати задачу в такому тестовому наборі, були згенеровані кілька зразків з моделей та перевірялися, чи проходить хоча б один з них юніт-тести (рис. 2.4).

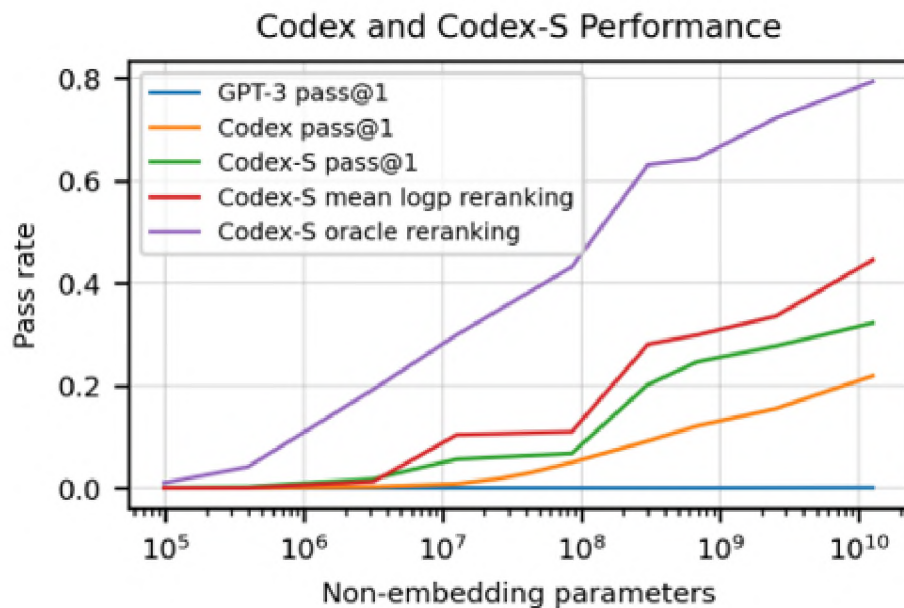


Рисунок 2.4 – Показники проходження дослідуваних моделей на наборі даних HumanEval як а функція розміру моделі [55]

При генерації лише одного зразка, модель Codex з 12 млрд параметрів (12B) розв'язує 28,8% цих задач, а Codex з 300 млн параметрів — 13,2%. У порівнянні, модель GPT-J з 6 млрд параметрів [56] досягає 11,4% на тому ж наборі даних, тоді як усі моделі GPT досягають близько 0%. Для покращення продуктивності моделі в задачі синтезу функцій з docstring, Codex тонко

налаштовували на автономні, правильно реалізовані функції. Результиуюча модель, Codex-S, вирішує 37,7% задач за допомогою одного зразка.

Генеративні моделі для створення коду переважно оцінюються на основі збігу зразків із еталонним рішенням, де збіг може бути точним або частковим (як у випадку з метрикою BLEU). Однак, у недавніх дослідженнях виявлено недоліки метрик на основі збігів для коду. Наприклад, [57] виявили, що BLEU має проблеми із захопленням семантичних особливостей, властивих коду, і запропонували кілька семантичних модифікацій до цієї метрики.

Більш фундаментально, метрики на основі збігів не можуть врахувати великий і складний простір програм, які функціонально еквівалентні еталонному рішенням. У результаті недавні роботи з неформального перекладу коду [58] та перекладу псевдокоду на код [59] звернулися до функціональної правильності, де зразок вважається правильним, якщо він проходить набір юніт-тестів. Дійшли висновку, що цю метрику слід застосовувати також для генерації коду на основі docstring.

Фреймворк, відомий як розробка, керована тестами, передбачає, що вимоги до програмного забезпечення перетворюються на тестові випадки перед початком реалізації, а успіх визначається програмою, яка проходить ці тести. Хоча небагато організацій повністю використовують розробку, керовану тестами, інтеграція нового коду зазвичай залежить від створення і проходження юніт-тестів.

Розробниками ШІ «Містраль» була презентована перша модель коду Codestral. Codestral – це генеративна модель з відкритою вагою, спеціально розроблена для створення коду. Він допомагає розробникам писати код і взаємодіяти з ним через загальну кінцеву точку API інструкцій та завершення. У міру того, як він освоює код та англійську мову, його використовувати для розробки передових програм штучного інтелекту для програмного забезпечення.

Codestral навчається на різноманітному наборі даних з 80+ мов програмування, зокрема найпопулярніші, такі як Python, Java, C, C++,

JavaScript і Bash. Він також добре працює на більш специфічних об'єктах, таких як Swift та Fortran. Ця широка мовна база гарантує, що Codestral може допомогти розробникам у різних середовищах програмування та проектах.

Codestral заощаджує час та зусилля розробників: він може завершити кодування функції, писати тести та завершувати будь-який частковий код, використовуючи механізм заповнення середини. Взаємодія з Codestral допоможе підвищити рівень програмування розробника та зменшити ризик помилок та багів.

## **Висновки до розділу 2**

Аналіз особливостей застосування фреймворків Angular, React, Svelte та Vue, а також їх переваг та недоліків показав, що універсальним рішенням для будь-якого проекту є React. Найкращим фреймворком для великих проектів є Angular. Для стартапів та невеликих проектів гарним вибором буде Vue. Якщо ж вам потрібна продуктивність, то варто зупинитись на Svelte.

Проведено два різних теста продуктивності фреймворків. Результати тесту з високим навантаженням показали, що майже однаково найшвидшими і найпродуктивнішими є Vue та Svelte. Angular та React показали гірші результати. В ході тестування з меншим навантаженням результати майже не відрізнялись. Зроблено висновок, що суттєва різниця в швидкодії обраних фреймворків починає проявлятися лише на великих обсягах даних та лише при деяких видах операцій.

Дослідження засосування різних моделей ШІ виявило низку обмежень, які включали труднощі з документаційними рядками, які описують довгі ланцюжки операцій, і з прив'язкою операцій до змінних. Однак, ідея генерації складних кодів показала безумовну перспективність і потребує подальших досліджень.

## РОЗДІЛ 3

### РЕЗУЛЬТАТИ ЗАСТОСУВАННЯ ОКРЕМИХ МОВНИХ МОДЕЛЕЙ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ ОПТИМІЗАЦІЇ КОДУ ВЕБДОДАТКУ

#### 3.1 Використання LLM GPT-4o для розпізнання та оптимізації JS коду

Кінцевим продуктом при веброзробці, з яким працюють як розробники, так і різноманітні аналітичні додатки, є програмний код, який написаний на обраній мові програмування. У практичній частині представлені результати проведеного експерименту, основною метою якого було дослідити можливості популярних великих мовних моделей LLM ШІ та перевірити гіпотези про можливість розпізнання, коригування й оптимізації програмного коду, а також можливість створення абсолютно оригінального коду із заданими параметрами як продукту штучного інтелекту.

Для тестування обрані закриті LLM (доступ через інтернет) – GPT4o, Gemini, а також локальні відкриті (open source) LLM, що встановлені на платформу LM Studio. Для експерименту був використаний js-файл з пакету завершеного вебсайту, який розміщено на хості, без вказання URL-адреси. Файл містив 148 рядків, у тому числі коментарі щодо призначення окремих функцій та методів обробки даних. Фрагмент коду представлено на рис. 3.1.

```

8  /*****/ // Check if module is in cache
9  /*****/ if(installedModules[moduleId]) {
10 /*****/   return installedModules[moduleId].exports;
11 /*****/ }
12 /*****/ // Create a new module (and put it into the cache)
13 /*****/ var module = installedModules[moduleId] = {
14 /*****/   i: moduleId,
15 /*****/   l: false,
16 /*****/   exports: {}
17 /*****/ };
18 /*****/
19 /*****/ // Execute the module function
20 /*****/ modules[moduleId].call(module.exports, module, module.exports, __webpack_require__)

```

Рисунок 3.1 – Фрагмент коду на мові JS з працюючого вебсайту

Вебсайт складався з класичного набору файлів, створених за технологіями HTML&CSS, а також мовою програмування JavaScript на фреймворку React. Інтерфейс користувача передбачав наявність окремих динамічних візуальних ефектів, наприклад, слайдер, масштабування зображень, внесення та елементарна обробка даних, а також забезпечення функціональності та адаптивності елементів сайту, наприклад, гамбургер-меню та інше.

Зазначений файл оброблявся в GPT-4o [60] від компанії OpenAI [53] з метою розпізнання та подальшої оптимізації. Код завантажувався безпосередньо в чат (ChatGPT) й аналізувався шляхом ведення діалогу, в ході якого використовувався сформований список запитань, наприклад:

1. Що це за код?
2. Цей код можна оптимізувати без втрати функціоналу?
3. Напиши власний код замість цього, щоб виконувався той же самий функціонал.
4. У твоєму коді є помилки?
5. Питання щодо довжини коду: твій коротший за початковий?

У ході експерименту отримані такі результати спілкування з GPT-4o.

Відповідь 1. Цей код є частиною проекту, який використовує Webpack для збирання модулів JavaScript. Він містить кілька скриптів для вебсторінки, включаючи функціонал для анімації елементів, роботи з якірними посиланнями, управління бургер-меню та надсилання форми. Давайте розглянемо основні компоненти:

1. Webpack Bootstrap: функція `_webpack_require_` використовується для завантаження модулів; кешування модулів: запобігає повторному завантаженню модулів, які вже були завантажені.

2. Модуль `anchors.js` – обробляє кліки на якірні посилання та плавно прокручує сторінку до відповідного елемента.

3. Модуль `animate.js` – додає клас `_active` до анімованих елементів при прокручуванні сторінки, щоб запускати анімацію.

4. Модуль `burger-menu.js` – керує відображенням бургер-меню та основного екрана, включаючи зміну `overflow` для блокування/розблокування прокручування сторінки.

5. Модуль `successful.js` – керує поведінкою форми та відображенням повідомлення про успішне відправлення, включаючи приховування форми та скидання значень полів.

6. Цей код використовується для створення інтерактивного та анімованого інтерфейсу користувача на веб-сторінці.

На рис. 3.2 наведено фрагмент висхідного коду, який описує бургер-меню, що точно був розпізнаний ШІ.

```

3  const burgerTrigger = document.querySelector('.burger'),
4     burgerMenu = document.querySelector('.burger-menu'),
5     mainScreen = document.querySelector('.main-screen'),
6     header = document.querySelector('.header'),
7     burgerMenuLink = document.querySelectorAll('.burger-menu__link');
8
9  function hide () {
10     mainScreen.classList.toggle('hide');
11     header.classList.toggle('hide');
12 }
13
14 burgerTrigger.addEventListener('click', () => {
15     burgerTrigger.classList.toggle('active');
16
17     if (burgerTrigger.classList.contains('active')) {
18         document.body.style.overflow = "hidden";
19     } else {
20         document.body.style.overflow = "";
21     }
22     hide();
23
24 });
25
26 for (let item of burgerMenuLink) {
27     item.addEventListener('click', (e) => {
28         if(item) {
29             burgerTrigger.classList.toggle('active');
30             document.body.style.overflow = "";
31             hide();
32         }
33     });
34 }

```

Рисунок 3.2 – Висхідний код бургер-меню

Відповідь 2. Оптимізація коду без втрати функціоналу може включати покращення продуктивності та читабельності. Ось деякі можливі покращення для представленого коду:

1. Використання `forEach` замість `for` для перебору елементів: `forEach` часто читається легше і більш декларативно.
2. Видалення непотрібних перевірок та операцій, наприклад, перевірка наявності елемента всередині циклу може бути виконана один раз до циклу.
3. Об'єднання загальних операцій на функції: створення допоміжних функцій для завдань, що повторюються.
4. Усунення дублювання коду: якщо та сама дія виконується у кількох місцях, його можна винести на окрему функцію.

Під час відповіді запропонований код (рис. 3.3), який можна порівняти з початковим (див. рис. 3.2).

```
// burger-menu.js
const burgerTrigger = document.querySelector('.burger'),
      burgerMenuLinks = document.querySelectorAll('.burger-menu__link');

function toggleBurgerMenu() {
  document.body.style.overflow = burgerTrigger.classList.contains('active') ? "hidden" : "";
  document.querySelector('.main-screen').classList.toggle('hide');
  document.querySelector('.header').classList.toggle('hide');
}

burgerTrigger.addEventListener('click', () => {
  burgerTrigger.classList.toggle('active');
  toggleBurgerMenu();
});

burgerMenuLinks.forEach(link => {
  link.addEventListener('click', () => {
    burgerTrigger.classList.toggle('active');
    document.body.style.overflow = "";
    toggleBurgerMenu();
  });
});
```

Рисунок 3.3 – Код бургер-меню, який був оптимізований GPT-4o

Основні зміни, які виявлені при порівнянні:

1. Використання `forEach` для перебору елементів: покращує читабельність і зручність коду.

2. Об'єднання загальних операцій: функція `toggleBurgerMenu` використовується для зміни стану бургер-меню.

3. Видалення надлишкових перевірок: спрощення умов більшої ясності.

Ці зміни покращують читабельність, підтримуваність та трохи підвищують продуктивність за рахунок зменшення дублювання коду та більш ефективного використання вбудованих методів.

Відповідь 3: Ось оптимізована версія вашого коду, яка виконує той самий функціонал (видано код, але не повний).

Відповідь 4 (про помилки): Мій попередній код не містить явних помилок, але є кілька уточнень та покращень, які можна внести для підвищення надійності та читання. Ось виправлений та доповнений варіант коду (рис. 3.4).

```
// burger-menu.js
const burgerTrigger = document.querySelector('.burger');
const burgerMenuLinks = document.querySelectorAll('.burger-menu__link');
const mainScreen = document.querySelector('.main-screen');
const header = document.querySelector('.header');

function toggleBurgerMenu() {
  document.body.style.overflow = burgerTrigger.classList.contains('active') ? "hidden" : "";
  mainScreen.classList.toggle('hide');
  header.classList.toggle('hide');
}

burgerTrigger.addEventListener('click', () => {
  burgerTrigger.classList.toggle('active');
  toggleBurgerMenu();
});

burgerMenuLinks.forEach(link => {
  link.addEventListener('click', () => {
    burgerTrigger.classList.toggle('active');
    document.body.style.overflow = "";
    toggleBurgerMenu();
  });
});
```

Рисунок 3.4 – Доповнений варіант коду після перевірки самим GPT-4o

Відповідь 5: Мій оптимізований код має ту ж функціональність, але за кількістю рядків він приблизно такий самий, як і вихідний. Основне покращення пов'язане з читабельністю та структуруванням коду. Оптимізація

не завжди означає скорочення кількості рядків, а скоріше покращення якості коду та його підтримуваності.

Таким чином, було отримано покращену версію коду зі змінами, наприклад, видалення зайвих перевірок та операцій, поєднання схожих операцій у функції, усунення дублювання коду та інші. Як результат, підвищено читабельність коду та його продуктивність, швидкість завантаження сторінки.

### 3.2 Використання мовних моделей ШІ Gemini та Codestral

Для розширення меж експерименту аналогічне дослідження відбувалось з Gemini від компанії Google [61], який можна назвати відповіддю на створення ChatGPT. Серед широкого спектру можливостей, пов'язаних із обробкою, генерацією текстів будь-якого змісту і складності, перекладу з різних мов та інших завдань, розробники декларують вміння писати код на мовах програмування Python, JavaScript, Java, C++, C#, PHP, Ruby, Swift. Діалог відбувався так само, як і в попередньому варіанті. Далі наведено зміст діалогу з чатом, у процесі якого відбувалося завантаження коду та «обговорення» із ШІ.

Питання 1: «Ти можеш проаналізувати код JavaScript?»

Відповідь 1: «Так, я можу проаналізувати код JavaScript декількома способами:

1. Статичний аналіз:

– Лістинг: можу перевірити ваш код на наявність поширених помилок та стилістичних проблем за допомогою таких інструментів, як ESLint та JSHint.

– Типова перевірка: можу перевірити типи даних ваших змінних та функцій, щоб переконатися, що вони використовуються правильно за допомогою таких інструментів, як TypeScript.

- Аналіз залежностей: можу проаналізувати ваш код, щоб побачити, які модулі він використовує та як вони пов'язані між собою.

- Покриття коду: можу підрахувати, скільки рядків коду виконується вашими тестами.

## 2. Динамічний аналіз:

- Налаштування: можу допомогти вам налагоджувати JavaScript, встановлюючи точки зупинки, переглядаючи значення змінних і виконуючи код покроково.

- Профіль: можу допомогти вам визначити, які частини вашого коду виконуються найдовше, щоб ви могли їх оптимізувати.

- Аналіз продуктивності: можу допомогти вам піддаватися вашому JavaScript-коду, який виконується в термінах використання пам'яті, CPU usage, і network requests.

Крім того, я можу:

- Пояснити код: можу надати докладні пояснення того, що робить ваш код.

- Запропонувати рефакторинг: можу запропонувати способи покращення вашого коду, зробивши його більш читаним, підтримуваним та ефективним.

- Згенерувати код: можу генерувати JavaScript code, що базується на ваших поданнях або examples.

Надалі було введено той самий код (див. рис. 3.1-3.2) для аналізу в Chat Gemini.

Завдання 1: «Проаналізуй код з погляду проблем ефективності та потенційних покращень».

Відповідь:

Наданий код є пакетом веб-пакета, що містить кілька файлів JavaScript з проекту. Він не написаний на чистому JavaScript, але використовує збирач, такий як webpack, для керування залежностями та їхнього об'єднання. Далі слідує опис розпізнання коду та можливості його оптимізації. ШІ визначає

Webpack Bootstrap як «систему модулів, яку використовує веб-пакет для керування залежностями між різними файлами JavaScript у проєкті».

Окремі файли Javascript були правильно класифіковані: Кожен із перерахованих файлів (anchors.js, animate.js, burger-menu.js, Success.js), ймовірно, буде містити функціональні можливості для певних частин веб-сайту, такі як плавне прокручування для прив'язок, анімація при прокручуванні, взаємодія з бургер-меню, та обробка успішного відправлення форми.

Завдання 2: «Оптимізуй вихідний код».

Відповідь 2: Оскільки я не маю доступу до окремих файлів JavaScript (anchors.js, animate.js, burger-menu.js, successful.js), я не можу безпосередньо оптимізувати наданий код. Однак я можу надати загальні рекомендації щодо оптимізації коду JavaScript.

Загальні методи оптимізації, які запропоновані Gemini:

1. Зменшення: зменшіть код, щоб видалити непотрібні пробіли, коментарі та форматування. Це може значно зменшити розмір пачки та скоротити час завантаження.

2. Розділення коду: розділіть великі пакети на менші блоки залежно від використання. Це може покращити початковий час завантаження сторінки та зменшити вплив окремих сценаріїв на продуктивність.

3. Літинг: використовуйте лінтер, як-от ESLint, для виявлення та виправлення потенційних помилок, забезпечення узгодженості стилю коду та покращення читабельності.

4. Маніпуляції DOM: мінімізуйте маніпуляції DOM за допомогою ефективних методів, таких як пакетне оновлення та віртуальні бібліотеки DOM.

5. Прослуховувачі подій: видаляйте прослуховувачі подій, коли вони більше не потрібні, щоб запобігти витокам пам'яті та підвищити продуктивність.

6. Сторонні бібліотеки: вибирайте добре оптимізовані сторонні бібліотеки та уникайте непотрібних залежностей.

7. Профілювання продуктивності. Використовуйте інструменти профілювання, щоб виявити вузькі місця продуктивності та оптимізувати відповідно.

Окрім загальних, Gemini видав конкретні оптимізації для кожного сценарію.

Для `anchors.js`:

- Оптимізуйте поведінку плавного прокручування, щоб мінімізувати вплив на продуктивність.

- Розгляньте можливість використання бібліотеки, як-от `smooth-scroll`, для ефективного прокручування.

Для `animate.js`:

- Використовуйте `IntersectionObserver` API, щоб спостерігати за елементами, що входять у вікно перегляду, і запускати анімацію лише за необхідності.

- Оптимізуйте обчислення анімації та використовуйте апаратне прискорення, якщо воно доступне.

Для `burger-menu.js`:

- Використовуйте переходи або анімацію CSS для плавного відкриття/закриття меню замість того, щоб покладатися виключно на JavaScript.

- Оптимізуйте обробку подій, щоб мінімізувати вплив на продуктивність.

Додаткові поради:

- Використовуйте AJAX для надсилання форм асинхронно без оновлення сторінки.

- Використовуйте переходи або анімацію CSS для повідомлення про успішне надсилання.

- Використовуйте CDN (Мережа доставки вмісту) для обслуговування файлів JavaScript із різних місць, зменшуючи затримку для користувачів у всьому світі.
- Увімкніть кешування файлів JavaScript, щоб зменшити кількість повторних завантажень.
- Використовуйте інструмент моніторингу продуктивності, щоб відстежувати продуктивність веб-сайту та визначати області, які потрібно покращити з часом.
- Дотримуючись цих вказівок і оптимізуючи кожен сценарій окремо, ви можете значно покращити продуктивність і загальну взаємодію з користувачем вашого веб-сайту.

Враховуючі зростання інтересу до локальних версій LLM, аналогічні дослідження проводились з Codestral-22B-v0.1-GGUF [62]. На відміну від розглянутий варіантів зазначений підхід має низку переваг, що пов'язано з кількома чинниками: конфіденційність даних; низька затримка та висока доступність; налаштування та адаптація під конкретні завдання; економічна ефективність; незалежність від зовнішніх послуг; підтримка локальних мов та особливостей. Для цього використовувалась платформа LM Studio [63]. Це інструмент із відкритим вихідним кодом, який оптимізує процес навчання, оцінки та використання сучасних мовних моделей локально. Як і GPT-4o, Codestral-22B-v0.1-GGUF видала прийнятний результат.

### **3.3 Оцінювання ефективності розробки вебдодатку за участю ШІ**

В процесі підготовки кваліфікаційної роботи була проведена серія експериментів з оптимізації програмного коду вебсайтів, а також протестовані моделі щодо написання ними програмного коду. Напрямок досліджень формує нові ідеї, однак, на основі отриманих результатів можна висунути припущення, що для різних випадків використання ШІ може бути економічно

обґрунтованою альтернативою для створення оптимізованого коду на етапі розробки (завершення, тестування) коду вебсайту порівняно з послугами комерційних маркетингових компаній, які проводять оптимізацію вебдодатків.

За даними різних джерел, наприклад [64], ціна на розробку вебсайту для середнього бізнесу стартує від 35 тис. грн (850\$ за курсом НБУ станом на 10.10.2024). При цьому розробка займає від 1 до 3 місяців. Розробка вебсайту включає комплекс операцій та забезпечення. Формула для розрахунку сумарної вартості вебсайту:

$$S_w = \sum_{k=1}^n w_k t, \quad (3.1)$$

де  $S_w$  – сумарна вартість робіт;

$w_k$  – вартість окремого виду роботи за одиницю часу;

$t$  – тривалість робіт;

$n$  – кількість видів робіт.

Тривалість і вартість передбачених робіт і витрат наведена в табл. 3.1 на основі даних, взятих з актуальних інтернет-джерел. Загальна вартість розробки вебсайту розрахована за формулою (3.1).

Таблиця 3.1 – Розцінки на окремі види робіт при розробленні вебсайту

Вид роботи/витрати	Вартість за од. часу	Затрати часу, одиниць	Сумарні витрати, грн
Доменне імя	400 грн/рік	1 рік	400
Хостинг	1200 грн/рік	1 рік	1200
CMS для сайту (безкоштовні версії)	0 грн/рік	1 рік	0
Технічне завдання (одноразово)	4500 грн	По факту	4500
Дизайн сайту (одноразово)	10000 грн	По факту	10000
Верстання сайту (за 5 сторінок)	9500 грн	По факту	9500
Програмування сайту	650 грн/год	40 год	26000
Наповнення сайту	350 грн/год	10 год	3500
Загальна вартість сайту для бізнес-компанії під ключ			55100

Час та вартість на тестування вебсайту перед передачею на хостинг знаходиться в межах 10-20 % часу, відведеного на програмування.

Використання ШІ для генерації коду вебсторінок стає все більш затребуваним завдяки здатності оптимізувати процеси розробки. Основні економічні переваги застосування ШІ в розробці вебсторінок можна розглянути в кількох ключових аспектах.

1. Зниження витрат на розробку: ШІ здатний автоматизувати процес написання коду, що зменшує обсяг ручної праці та скорочує час, необхідний для виконання рутинних завдань. Це дозволяє компаніям економити на витратах, пов'язаних з наймом додаткових програмістів. Наприклад, ШІ може генерувати HTML, CSS або JavaScript код для типових компонентів, таких як кнопки, форми чи інтерактивні елементи, що суттєво зменшує робоче навантаження. З огляду на розцінки, економічний ефекти на розробці може скласти від 20% на програмуванні, тобто 10120 грн.

2. Підвищення продуктивності та швидкості розробки. Інструменти ШІ дозволяють розробникам працювати швидше, оскільки вони можуть генерувати готові фрагменти коду на основі коротких запитів або описів функцій. Це означає, що проекти завершуються швидше, що важливо для компаній, які хочуть швидше виводити свої продукти на ринок. Швидкість розробки особливо критична в галузі, де нові продукти і функції повинні швидко відповідати на зміни в потребах користувачів та ринкові тенденції.

3. Оптимізація якості коду: ШІ може використовуватися для перевірки коду на помилки, дотримання стандартів або оптимізацію продуктивності, що знижує витрати на тестування та покращує якість кінцевого продукту. Наприклад, інструменти ШІ можуть автоматично визначати можливі помилки, пропонувати виправлення або покращення, що зменшує ризики, пов'язані з багами або низькою продуктивністю веб-сторінок. Це особливо важливо в умовах, де стабільність та швидкість роботи вебресурсу впливають на конверсію та утримання клієнтів. Оптимізація займає від 10 % від вартості розробки, тобто 2600 грн.

Таким чином, навіть часткове залучення ШІ дозволяє отримати економію коштів  $SUM = 10120 + 2600 = 12720$  грн.

По відношенню до вартості цілого проекту економія складе:

$$12720/55100 \times 100 = 23,08 \%$$

Окрім фінансових вигод розглядаються також і організаційні переваги.

4. Економія часу на навчання та адаптацію нових співробітників: використання ШІ для генерації коду дозволяє менше покладатися на досвідчених програмістів для виконання базових завдань. Це корисно для новачків, яким ШІ може допомогти генерувати код, адаптуватися до нових технологій та стандартів без великої потреби в тривалому навчанні. В результаті компанії можуть швидше інтегрувати нових співробітників у робочий процес, знижуючи витрати на їхнє навчання.

5. Інновації та конкурентна перевага: використання ШІ дозволяє компаніям тестувати нові ідеї без значних інвестицій в ресурси, а також бути більш гнучкими до змін. ШІ може швидко адаптувати існуючий код або генерувати нові ідеї на основі поточних трендів, що допомагає компаніям залишатися на передових позиціях на ринку. Це особливо важливо в висококонкурентних галузях, таких як електронна комерція та онлайн-сервіси, де постійні інновації є запорукою успіху.

6. Ефективне використання хмарних технологій та масштабованість: інтеграція ШІ для генерації коду в хмарних платформах дозволяє компаніям автоматизувати процес розробки у великому масштабі. Наприклад, для стартапів або невеликих компаній ШІ може забезпечити доступ до високотехнологічних рішень без необхідності великих інвестицій в інфраструктуру або кадрові ресурси.

Таким чином, застосування штучного інтелекту при генерації коду для вебдодатків може значно підвищити ефективність розробки, знизити витрати, прискорити вихід на ринок та покращити якість продукту. В умовах високої конкуренції та швидких технологічних змін ці економічні переваги стають вирішальними для компаній, які прагнуть досягти успіху в цифровому світі.

### **Висновки до розділу 3**

У результаті роботи над останнім розділом були отримані результати експерименту з використання мовних моделей ШІ для оптимізації і написання коду згідно зразка або визначеного завдання.

Отримані результати з досліджених мовних моделей ШІ показують здатність до точного розпізнання коду, написаного на JavaScript, особливостей функціоналу, а також внесення пропозицій щодо оптимізації, раціоналізації коду без втрати функціоналу. Всі моделі працювали через завантаження досліджуваного коду безпосередньо в чат. Найбільш розлогі рекомендації щодо оптимізації видала система Gemini. Разом із тим, при спілкуванні з Gemini не вдалось отримати генерацію коду, тобто все звелось до узагальнених положень. Швидку генерацію нового коду надав GPT 4.0.

Проведено економічний аналіз, порахована орієнтовна вартість реалізації розробки вебдодатку для середнього бізнесу. За оцінками вартість такого проєкту 52100 грн. Застосування штучного інтелекту при генерації коду для вебсторінок може значно підвищити ефективність розробки, знизити витрати (за оцінками від 12700 грн або 23,08 %), прискорити вихід на ринок та покращити якість продукту.

Результати дослідження опубліковані в статті. Повний текст публікацій представлено в додатку Г.

## ВИСНОВКИ

Вимоги до вебдодатків наростають надзвичайно швидко. Вони стають не просто елементом, а важливою складовою успішного бізнесу. В ході виконання кваліфікаційної роботи було проаналізовано значну кількість різноманітного інструментарію веброзробника, який впливає на якість та оптимізацію вебдодатків. Підсумовуючи результати проведеного дослідження, яке стосувалося пошуку можливих шляхів технічної оптимізації програмних кодів, які постають перед вебпрорамистами при розробці та застосуванні веб-додатків на JavaScript, можна сформулювати наступні висновки.

Аналіз особливостей застосування фреймворків Angular, React, Svelte та Vue, а також їх переваг та недоліків показав, що найбільш універсальним рішенням для будь-якого проєкту є React. Найкращим фреймворком для великих проєктів є Angular. Для стартапів та невеликих проєктів гарним вибором буде Vue. Якщо ж вам потрібна продуктивність, то варто зупинитись на Svelte.

Результати з досліджених LLM показують здатність до точного розпізнання коду, написаного на JavaScript, особливостей функціоналу, а також внесення пропозицій щодо оптимізації, раціоналізації коду без втрати функціоналу. Всі моделі працювали через завантаження досліджуваного коду безпосередньо в чат. Найбільш розлогі рекомендації щодо покращення отримано від Gemini. Швидку генерацію нового коду надав GPT-4o.

Подальшими напрямками досліджень можуть бути нові версії open source LLM, а також спеціалізовані платформи, призначені для генерації програмних кодів на інших мовах програмування.

Проведено економічний аналіз, порахована орієнтовна вартість реалізації розробки вебдодатку для середнього бізнесу. А основі оцінок визначено, що застосування штучного інтелекту при генерації коду для вебсторінок може значно підвищити ефективність розробки, знизити витрати

(за оцінками від 12700 грн або 23,08 %), прискорити вихід на ринок та покращити якість продукту.

Запропонована методика досліджень може бути корисною для оптимізації кодів більшої складності з мінімальними витратами часових і людських ресурсів.