Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента *Єфременко Дмитра Костянтиновича*
(ПІБ)

академічної групи *121М-19-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

на тему: *Дослідження взаємодії фреймворків та програмної платформи Node.js з метою розробки високо динамічних веб-застосунків з використанням Javascript*

_____ *Д.К.Єфременко*

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
розділів кваліфікаційної роботи				
спеціальний	Проф. Алексеєв М.О.			
економічний	Доц. Касьяненко Л.В.			

Рецензент				

Нормоконтролер	Доц. Сироткіна О.І.			

Дніпро
2020

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
Завідувач кафедри
Програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 20 20 Року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

студенту *121М-19-1* Єфременко Дмитру Костянтиновичу
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи *Дослідження взаємодії фреймворків та програмної платформи Node.js з метою розробки високо динамічних веб-застосунків з використанням Javascript*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 22.10.2020 р. № 888-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт дослідження – процеси проєктування, написання і використання вебзастосунків та вебсайтів у сучасному програмуванні з використанням фреймворків і програмної платформи Node.js.

Предмет дослідження – методи ефективного використання фреймворків і програмної платформи Node.js.

Мета кваліфікаційної роботи – ефективне користування початківцями можливими інструментами і шляхами розробки вебзастосунків, досягнуте шляхом інформування про новітні тенденції, фреймворки та методи їх використання.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна результатів кваліфікаційної роботи полягає в удосконаленні сучасних методів динамічної обробки URL посилань з використанням мови програмування Javascript на платформі Node.js.

Практична цінність – результати проведеного дослідження і розроблений застосунок можуть бути використані розробниками, аспірантами та студентами які цікавляться або спеціалізуються у сфері програмування та тестування клієнтської

та/або серверної частин вебзастосунків, серверів чи вебсайтів з використанням мови програмування Javascript на платформі Node.js.

4 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець**
Аналіз теми та постановка задачі	12.09.2020-30.09.2020
Збір, систематизація та дослідження інформації про фреймворки та їх взаємодію з програмною платформою Node.js.	01.10.2020-31.10.2020
Розробка вебзастосунку для динамічної обробки і скорочення URL посилань та аналітики трафіку	01.11.2020-08.12.2020

Завдання видав _____ *Алексеєв М.О.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Єфременко Д.К.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *12.09.2020 р.*

Термін подання кваліфікаційної роботи до ЕК *10.12.2020*

РЕФЕРАТ

Пояснювальна записка: 104 стор., 24 рис., 6 таблиць, 3 додатки, 27 джерел.

Об’єкт дослідження – процеси проєктування, написання і використання вебзастосунків та вебсайтів у сучасному програмуванні з використанням фреймворків і програмної платформи Node.js.

Предмет дослідження – методи ефективного використання фреймворків і програмної платформи Node.js.

Мета кваліфікаційної роботи – ефективне користування початківцями можливими інструментами і шляхами розробки вебзастосунків, досягнуте шляхом інформування про новітні тенденції, фреймворки та методи їх використання.

Методи дослідження – у процесі дослідження були використані теоретичні та емпіричні наукові методи: опису, аналізу, абстрагування, узагальнення на основі даних, класифікації і пояснення.

Наукова новизна результатів кваліфікаційної роботи полягає в удосконаленні сучасних методів динамічної обробки URL посилань з використанням мови програмування Javascript на платформі Node.js.

Практична цінність – результати проведеного дослідження і розроблений застосунок можуть бути використані розробниками, аспірантами та студентами які цікавляться або спеціалізуються у сфері програмування та тестування клієнтської та/або серверної частин вебзастосунків, серверів чи вебсайтів з використанням мови програмування Javascript на платформі Node.js.

Список ключових слів: Node.js, фреймворки, full-stack розробка, сервер, клієнт

ABSTRACT

Explanatory note: 104 pages, 24 figures, 6 tables, 3 applications, 27 sources.

Object of research: processes of designing, writing and using web applications and websites in modern programming using frameworks and software platform Node.js.

Subject of research: efficient use methods of frameworks and software platform Node.js.

Purpose of master's thesis: efficient use of possible tools and ways to develop web applications by beginners achieved with informing about the latest trends, frameworks and methods of their use.

Research methods: both theoretical and empirical scientific methods were used in the research conduct: description, analysis, abstraction, generalization based on data, classification and explanation.

Originality of research is associated with the improvement of modern methods of dynamic processing of URL using the Javascript programming language on the Node.js platform.

Practical value of the results is that they can be used by developers, graduates and students interested or specializing in programming and testing frontend and/or backend parts of web applications, servers and websites using the Javascript programming language on the Node.js platform basis.

Keywords: Node.js, frameworks, full-stack development, backend, frontend

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – База даних;

ПЗ – Програмне забезпечення;

ООП – Об’єктно орієнтований підхід;

URL – Uniform Resource Locator – адреса ресурсу;

TCP – Transmission Control Protocol – протокол керування передачею

IP – Internet Protocol – мережевий протокол;

HTTP – Hyper Text Transfer Protocol – протокол передачі гіпертексту;

FTP – File Transfer Protocol – протокол передачі файлів;

HTML – Hyper Text Markup Language – мова розмітки гіпертекстових документів;

JS – Javascript;

CSS – Cascading Style Sheets – каскадні таблиці стилів;

JSON – Javascript Object Notation;

MVC – Model-View-Controller;

SQL – Structured query language;

API – Application programming interface – прикладний програмний інтерфейс;

MERN – MongoDB, Express, React, NodeJS;

NPM – Node package manager;

ODM – Object data modeling;

DOM – Document object model;

WWW – World Wide Web;

CPU – Central processing unit.

ВСТУП

Актуальність дослідження. За наявності великого різноманіття систем з різними властивостями і потребами існує необхідність у зборі та аналізі інформації про наявні сучасні засоби розробки. Знання про існуючі фреймворки, бібліотеки та можливості їх взаємодії дає змогу правильно підібрати інструменти для розробки конкретних проєктів, що позитивно впливає на побудову злагодженої архітектури. Вона слугує програмістам у подальшій розробці, неодмінно зменшує вартість розроблюваних систем і їх подальшу підтримку, а також істотно знижує ризик необхідності перебудови системи при масштабуванні.

В усіх різновидах операційних систем, в усіх браузерах, на всіх видах пристроїв підтримується скриптова інтерпретована мова – Javascript. Згідно статистики вона використовується у 95% вебзастосунків. Важливою властивістю є те, що такі застосунки можуть працювати без встановлення будь-яких виконуваних файлів на комп’ютер користувача. Таким чином, якщо ви хочете, щоб створеним застосунком міг користуватися будь-хто без додаткових дій – то Javascript майже єдина альтернатива. Javascript – найпопулярніша мова для розробки клієнтської частини веб ресурсів.

Початково створений для більш вузьких задач, Javascript, завдяки еволюції веб в сторону інтерактивних вебзастосунків, отримав цілу інфраструктуру фреймворків, бібліотек, протоколів і методів. Також з’явилося платформа для виконання клієнтських і серверних застосунків.

Ця програмна платформа називається Node.js. В її основу автор Райан Дал заклав подійно-орієнтований підхід. З ростом зацікавленості інтернет користувачів соціальними мережами та іншими інтерактивними вебсайтами – різко зросла популярність Node.js, як платформи для створення застосунків, що швидко реагують на дії користувачів: чатів, ігор, інструментів спільної роботи, новин, блогів, комерційних та інших ресурсів. Далі наводиться перелік інформаційних гігантів, які використовують Node.js у розробці та підтримці

своїх онлайн майданчиків у мережі: Reddit, Medium, Wikipedia, eBay, New Yorker, Yahoo, PayPal, NASA, Groupon, Wall Street Journal, Klout, Secret, Opencare, Shutterstock і т.д. "Навколо Node.js побудована сучасна екосистема, і ми отримуємо від неї численні переваги", – сказав інженер eBay Сентіл Падманабхан. «Інструменти розробки, бібліотеки тестування і автоматизації та багато інших корисних компонентів розповсюджуються через NPM, тому базові знання роботи з Node.js є невід'ємними для будь-якого веброзробника.

У даній кваліфікаційній роботі ми будемо систематизувати інформацію з відкритих джерел про фреймворки, програмну платформу Node.js та ефективні можливості їх взаємного використання. Практичними прикладами отриманих результатів виступають знання про наявні тенденції розвитку і набори інструментів розробки, а також створена програма – вебресурс для організації обробки та скорочення URL посилань реалізований мовою програмування Javascript на основі платформи Node.js.

Мета кваліфікаційної роботи – ефективне користування початківцями можливими інструментами і шляхами розробки вебзастосунків за допомогою інформування про новітні тенденції, фреймворки та методи їх використання.

Завдання дослідження – для досягнення поставленої мети в роботі сформульовані і вирішені такі завдання:

1. Проаналізувати найновіші тенденції розвитку розробки вебзастосунків та сайтів
2. Зібрати, систематизувати та дослідити інформацію про фреймворки та програмну платформу Node.js
3. Розробити вебресурс для організації обробки та скорочення URL посилань мовою програмування Javascript з метою демонстрації можливостей правильно підібраного набору інструментів

Об'єкт дослідження – процеси проєктування, написання і використання вебзастосунків та вебсайтів у сучасному програмуванні з використанням фреймворків і програмної платформи Node.js.

Предмет дослідження – методи ефективного використання фреймворків і програмної платформи Node.js.

Методи дослідження – у процесі дослідження були використані теоретичні та емпіричні наукові методи: опису, аналізу, абстрагування, узагальнення на основі даних, класифікації і пояснення.

Наукова новизна результатів кваліфікаційної роботи полягає в удосконаленні сучасних методів динамічної обробки URL посилань з використанням мови програмування Javascript на платформі Node.js.

Практична цінність – результати проведеного дослідження і розроблений застосунок можуть бути використані розробниками, аспірантами та студентами які цікавляться або спеціалізуються у сфері програмування та тестування клієнтської та/або серверної частин вебзастосунків, серверів чи вебсайтів з використанням мови програмування Javascript на платформі Node.js.

Особистий внесок автора:

1. Наукові результати роботи отримані автором самостійно.
2. Вибір методів досліджень і технологій реалізації.
3. Розробка теоретичної частини роботи, що збирає, систематизує і досліджує інформацію про існуючі тенденції розробки ПЗ для веб.
4. Розробка веб застосунку «SL1nk» з використанням набору інструментів розробки MERN на основі програмної платформи Node.js.
5. Оцінка отриманих результатів.

Структура та обсяг роботи. Робота складається зі вступу, чотирьох розділів та висновків. Містить 104 сторінки, у тому числі 75 сторінок тексту основної частини з 24 рисунками, список використаних джерел із 27 пунктами на 3 сторінках, 3 додатки на 15 сторінках

РОЗДІЛ 1
АНАЛІЗ ТЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Використання Javascript та інших інструментів веброзробки у створенні вебзастосунків

Веброзробка – один з найпростіших і тому популярних напрямків серед початківців програмістів. Для роботи у будь-якому текстовому редакторі і браузері не обов'язково вивчати алгоритми на високому рівні, результат кожного етапу написання програм очевидний. Для допомоги є величезний набір інструментів, і немає потреби на початку знати як все влаштовано. Достатньо розуміти як досягти результату за допомогою інструментів та конструкцій мови програмування, як то фреймворки, а вони, у свою чергу, дбають про те, щоб все працювало як слід. В той же час навіть побіжний огляд вакансій на StackOverflow доводить, що сучасні IT-компанії використовують велике розмаїття наборів інструментів і технологій та потребують спеціалістів з поглибленими знаннями.

Знання Javascript вважається ключовою навичкою в контексті веброзробки. Скрипти Javascript – це програми, які працюють з об'єктами HTML-документа. Ви можете змінити або навіть замінити весь завантажений документ HTML іншим. За допомогою сценаріїв Javascript можливо представити їх як об'єкти, клієнтське програмування.

Вікно браузера відповідає об'єкту вікна, а документ HTML, завантажений у вікно, відповідає об'єкту документа. Об'єкт документа є частиною вікна. Елементи документа HTML відповідають об'єктам, які є частиною документа. Весь набір має ієрархічну структуру, яка називається об'єктною моделлю документа, або DOM. Об'єкт являє собою контейнер для зберігання інформації. Він характеризується властивостями, методами і подіями, на які він може реагувати.

Мова програмування Javascript призначена для створення інтерактивних HTML-документів. Це об'єктно-орієнтована мова для розробки вбудованих

додатків, які працюють як на стороні клієнта, так і на стороні сервера. Синтаксис мови аналогічний синтаксису мови Java, тому його часто називають Java-подібним. Клієнтські програми запускаються браузером, переглядають вебдокументи на машині користувача, серверні застосунки працюють на сервері.

Обидва типи застосунків використовують загальний мовний компонент який називається ядром. Він включає визначення стандартних об'єктів і структур (змінних, функцій, основних об'єктів і інструментів для взаємодії з Java-аплетами), а також відповідні компоненти мовних надбудов, які містять всі типи додатків для визначення об'єктів.

Конструкції сценаріїв Javascript вбудовуються в HTML-сторінки і інтерпретуються в машинний код вбудованим у браузер інтерпретатором, що виконується компьютером при завантаженні сторінок чи певних діях з боку користувача або сервера. Призначення клієнтських скриптів Javascript:

- Швидка перевірка заповнених користувачем полів HTML-форм перед їх відправкою на сервер. Скрипти Javascript можуть обробляти дані, введені користувачами в поля форми, а також події, які відбуваються під час маніпуляції користувачем з мишею, копіювати інші HTML-сторінки у вікно браузера або скасовувати вже завантажені сторінки.
- Створення динамічних HTML-сторінок разом з каскадними таблицями стилів і об'єктною моделлю документа.
- Взаємодія з користувачем при вирішенні "локальних" завдань. Зокрема, скрипти Javascript широко використовуються для створення різних візуальних ефектів. Наприклад, зміна зовнішнього вигляду елементів управління, на які наведений курсор миші, анімація графічних зображень, створення звукових ефектів і т.д. Механізм локальних файлів дозволяє сценаріям Javascript зберегти локальну інформацію, введену користувачем на комп'ютері.

Асинхронність – одна з чудових особливостей Javascript, хоча вона створює розкол між розробниками: комусь це подобається, комусь ні. Необхідно зрозуміти плюси і мінуси цієї технології. У синхронних програмах, якщо у нас є

два рядки коду (Рядок 2 після Рядка 1), то Рядок 2 не може працювати до тих пір, поки Рядок 1 не завершить своє виконання. В асинхронних програмах, ми можемо мати два рядки коду (Рядок 2 після Рядка 1), де Рядок 1 містить команди, які будуть виконуватися у майбутньому, а Рядок 2 працює до завершення команди в Рядку 1. Принцип синхронного коду можна порівняти з роботою залізничної каси – ви знаходитесь в черзі людей, які чекають, щоб купити собі квитки на потяг. Ви стоїте в черзі і не можете купити квиток на потяг, поки ваша черга не настане. Так само і люди за вами не зможуть купити квитки, поки ви не закінчили. А принцип асинхронного коду можна порівняти з обслуговуванням у ресторані: люди можуть замовляти їжу у той же час як і ви замовляєте щось собі. Вам не потрібно чекати, поки люди до вас завершать своє замовлення, щоб зробити те, щоб вам потрібно, якщо ваше замовлення виконується швидше, і навпаки. Всі отримають свої замовлення по мірі їх виконання.

Послідовність, з якою люди отримують їжу часто корелює з послідовністю, в якій вони замовили їжу, але ці послідовності не завжди повинні бути ідентичні. Наприклад, якщо ви замовляєте стейк, після чого хтось замовив склянку води, то, ймовірно, він отримає своє замовлення першим, тому що принести стакан води не займає так багато часу, як приготувати стейк.

Якщо розглянути принципи потоковості – можна зробити висновок, що асинхронний – не означає одночасний чи багатопотоковий. Javascript має асинхронний характер, але він однопотоковий. Це наче в кафе з одним працівником, що робить все – готує і подає. Але, якщо цей працівник працює швидко і може переключатися між завданнями ефективно, то може здаватися, що у кафе працюють кілька робітників.

Щоб створити синхронну послідовність операцій – Javascript код потрібно структурувати по іншому і найпростіший спосіб це зробити – використати callback функції або функції зворотнього виклику. В Javascript вони активно використовуються. Майже всі вебзастосунки використовують зворотній виклик для обробки подій, як то: window.onclick, setTimeout та setInterval, а також для асинхронних запитів (AJAX) та ін. В розробці існує ряд завдань, де кожен крок

залежить від результатів попереднього кроку. Не важко отримати таке використовуючи синхронні мови програмування. При спробі зробити це в асинхронному коді, дуже легко отримати так зване callback-пекло – розповсюджену проблему, коли кількість вкладених одна в одну функцій зворотнього виклику стає надмірно великою. Сервери на платформі Node.js і код з великою кількістю AJAX-запитів часто страждають від цієї проблеми. Такий код важко читати, а спроба реорганізувати його, коли потрібно внести зміни, ускладнює та сповільнює виконання роботи. Глибоко "заплутані" функції зворотнього виклику в коді – це знак, що коду необхідна реорганізація. Існує кілька різних стратегій для цього. Promises (укр. “обіцянка”) – популярний спосіб позбавлення від надмірної кількості зворотніх викликів. Ідея полягає в тому, що замість того, щоб використовувати функції, які приймають вхідні дані і зворотні виклики, створити функцію, що повертає об'єкт promise, тобто, об'єкт, що представляє значення, яке буде існувати в майбутньому. Спочатку, це були конструкції, представлені такими бібліотеками як Q і when.js, але ці конструкції стали настільки популярними, що Promises стали стандартом в ECMAScript 6.

Варто відзначити відкритість мови Javascript. Вона має велику кількість бібліотек і фреймворків, що дозволяє легко вирішувати завдання будь-якого типу. Можна писати комп’ютерні і мобільні додатки з використанням платформи Electron та NativeScript або React Native відповідно.

Для початку роботи потрібно знати базові концепції Javascript та веброзробки і програмування в цілому:

- Об'єктно-орієнтована JS – конструктори, об’єкти і спадкування.
- Функціонал JS – функції вищого порядку, схема, рекурсія.
- Основи HTML, CSS, JQuery.
- Git – система контролю версій. Необхідний інструмент для розробників для управління файлами на серверах, співпраці з командою, поширення коду і уникнення розбіжностей при редагуванні.

Наразі Git, яку створив Лінус Торвальдс, є найбільш визнаним інструментом контролю версій так само хостинг кодів GitHub, що базується на Git. Ось основні необхідні навички:

- Створення і переміщення файлів в каталогах.
- Ініціалізація і фіксація в Git.
- Настроювання репозиторіїв в GitHub.

Також можливе використання альтернативного сервісу – GitLab. Запущений у 2011 році Дмитром Запорожцем, GitLab був заточений під командну роботу, тоді як GitHub більш зручний для індивідуальної роботи. Основні переваги GitHub з погляду бізнес-керівників полягають у великій кількості доступних користувачів у системі. Також чимало проєктів з відкритим вихідним кодом розміщується саме тут. Наявність безкоштовних командних приватних репозиторіїв часто стає головним фактором вибору GitHub. З погляду розміщення пропрієтарної інформації, персональних даних, іншої чутливої інформації, безперечно, перевага на боці GitLab, тому що систему можна встановити на власних серверах. З погляду керування задачами у проєкті (issue management) GitLab допоможе вимірювати та відстежувати повний життєвий цикл розробки – від планування до розгортання. Найбільш важливими є такі засоби, як Time Tracking, Burndown Charts, Issue Due Dates, функція переміщення задачі в інший проєкт. Також у системі контролю версій GitLab є змога робити імпорт та експорт з інших систем керування (GitHub, Bitbucket, Google Code, FogBugz, Gitea) або з будь-якого доступного Git URL. Окрім цього, існує підтримка різних видів репозиторіїв: Mono Repos, Conan (C++), Go, Composer (PHP), PyPI (Python), RPM та Debian (Linux). А ось GitHub краще підтримуватиме Visual Studio від Microsoft, тому що саме ця компанія зараз володіє сервісом GitHub. Клієнти, що зосереджені на інтелектуальній власності коду, часто віддають перевагу GitLab, адже система може бути розміщена на приватних серверах. Недоліком таких рішень є залежність від клієнта і розробників в питаннях конфігурації і оновлення версій системи. Для тих компаній, що прагнуть мінімізувати свою відповідальність за обробку персональних даних користувачів і довіряють

корпораціям, найкращим рішенням є GitHub, де не потрібно турбуватися про інфраструктуру і можна легко контролювати доступ і рівень адміністративних прав розробників.

Бази даних, схеми, моделі і ORM – одні з найважливіших елементів веброзробки. Якщо в застосунку необхідно завантажувати або зберігати будь-які дані, які б не втрачалися при оновленні сторінки – необхідно використовувати базу даних. Потрібно розрізняти реляційні і нереляційні бази даних і розуміти типи з'єднань. Також необхідно знати SQL і познайомитись із різними системами управління базами даних. Уміння працювати з ORM теж важливе. Також корисно вивчати популярні бібліотеки (наприклад GSAP, Bootstrap) і препроцесори CSS, як Sass, що допоможе писати на CSS ефективно.

HTML і CSS є основою будь-якого веброзробника. Іноді веброзробникам не обов'язково знати їх досконало, але вони повинні їх розуміти. Щоб спростити роботу з HTML, ми можемо обрати один з популярних шаблонів, наприклад pug, JQuery для маніпуляції з DOM. Створюючи зовнішній вигляд сторінки за допомогою HTML і CSS, можливе використання трансляторів подій і бібліотеку JQuery для управління DOM. Оскільки різні браузери трохи різняться за поведінкою Javascript і реалізацією об'єктної моделі документа, потрібні налаштування до кожного браузера, якщо наші вебзастосунки націлені на нього. Оскільки Javascript є нетипізованим інтерпретатором і може працювати в різних середовищах, кожна з яких має свої власні міркування сумісності. Тому програміст повинен бути дуже обережним, щоб переконатися, що його код працює належним чином в найрізноманітніших можливих конфігураціях. Інтерпретатор аналізує кожен блок скрипта окремо.

Синтаксичні помилки Javascript і HTML легше знайти, якщо зберігати код в окремими логічними блоками або використовувати безліч невеликих пакетних файлів.

На відміну від більшості мов, Javascript не керується тільки концепціями введення (input) і виведення (output). Він спроєктований таким чином, щоб запускатися як мова сценаріїв, вбудованих в середовище виконання.

Найпопулярніше середовище виконання це браузер, проте інтерпретатори Javascript присутні і в Adobe Acrobat, Photoshop, Yahoo Widget, і в серверному оточенні Node.js.

1.2. Виникнення платформи Node.js як фреймворка Javascript. Принципи її побудови. Сильні та слабкі сторони

Ідея створення Node.js була обумовлена об'єктивними причинами. Райан Дал у 2008р. працював з модулем Nginx – одного з найпродуктивніших серверів. Продуктивність була пов'язана з його устроєм – він працює асинхронно, що вимагає не блокуючого вводу-виводу. На той час більшість серверів використовували просту багатопоточну модель. Потік або ж процес – це самодостатній набір інструкцій, який операційна система може запланувати та виконати на ядрі процесора. Більшість вебсерверів та застосунків використовують таку модель коли для кожного з'єднання виділяється один потік (або ж процес, або ядро). Щоб обробити декілька запитів одночасно сервер для кожного запиту створює новий потік для його обробки. Це звичайна клієнт-серверна архітектура, в якій клієнт запитує потрібний ресурс у сервера і той відправляє ресурс у відповідь та, відповівши, перериває з'єднання. Сервер обробляє декілька запитів, але по одному на потік (така модель називається thread-per-request model). Припустимо у сервера 4 ядра, отримавши 4 запита він обробляє їх. Як тільки з'явиться 4+1 запит його обробка не відбудеться – він буде чекати поки не звільниться якийсь з чотирьох потоків (рис. 1.1).

Це можна порівняти з грою в шахи, де набір інструкцій для обробки запиту – правила, а кожна HTTP транзакція – шахова партія, з одного боку шахівниці вебсервер – гросмейстер, з іншого – віддалений клієнт, браузер. Кожна партія грається до кінця (з'єднання, процес і потік зв'язані) і весь цей час процес, що виконується на сервері залишається заблокованим в очікуванні сигналу – наступного ходу від клієнта. Рішенням проти таких обмежень стає збільшення

ресурсів (пам’яті, ядер тощо), використання сторонніх модулів (нових «правил»).

Така система погано масштабується. Із зростанням до тисяч одночасних з’єднань сильно падає продуктивність. Ця проблема у 2000-х отримала назву – C10k Problem (10k connections – проблема 10 тисяч з’єднань). На сьогодні

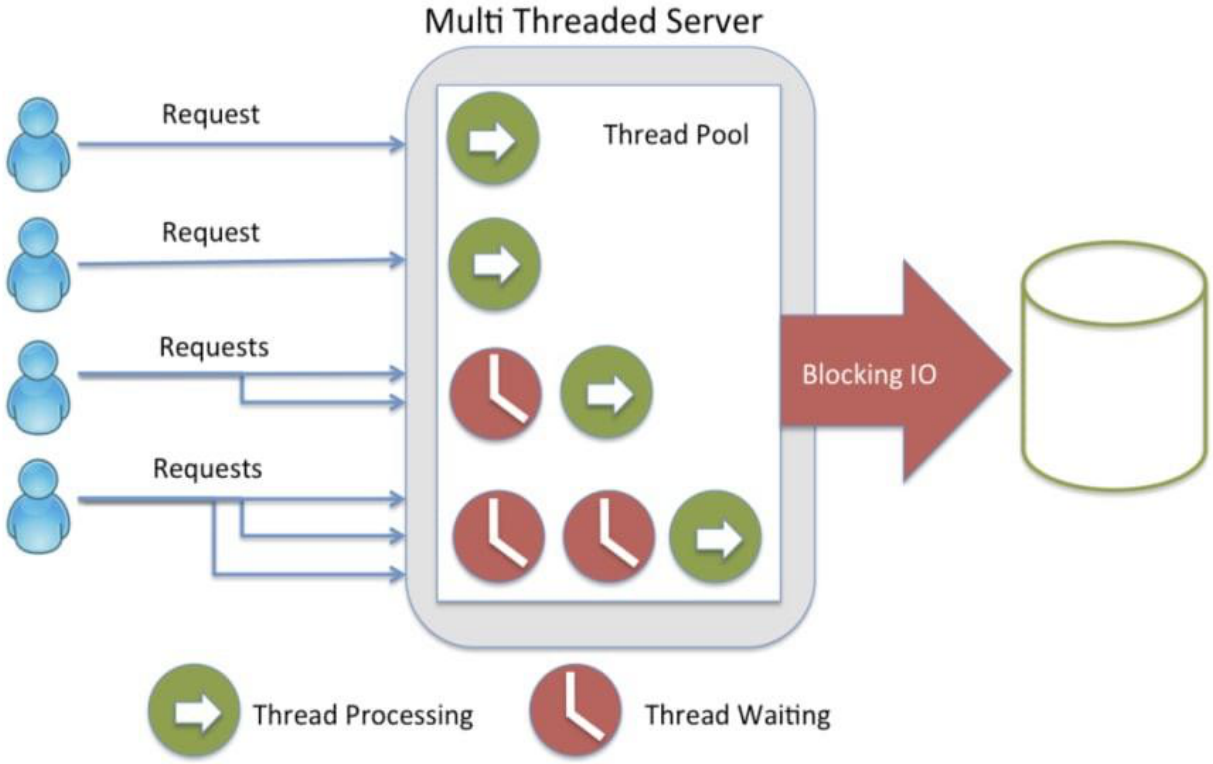

Рис.1.1. Схема багатопоточного серверу

сервери та клієнтські комп’ютери працюють швидше та ефективніше, але проблема не вичерпалася. Для 10 мільйонів клієнтських з’єднань на одній машині проблема повертається, і називається тепер як C10M Problem.

У NGINX робочий процес або потік ніколи не блокується при обробці мережевого трафіку. Він подібний до гросмейстера, який грає «сеанс одночасної гри» – на багатьох дошках із багатьма гравцями (клієнтами) одночасно. Зробивши хід, процес переміщується до інших дощок, на яких гравці чекають на відповідь. На практиці кожне нове з’єднання створює файловий дескриптор і споживає невеликий об’єм пам’яті в робочому процесі, таким чином процеси NGINX можуть залишатися прив’язаними до конкретних ядер процесора та

приймати нові запити. Надихнувшись NGINX та ще одним проєктом – Rack, що був спрощеною абстракцією сервера, перетворюючи вебсервер на інтерфейс з єдиною функцією, де можна робити запити і отримувати відповіді. А також, вивчаючи новий рушій Javascript – V8 від Chrome, Райан Дал захотів розробити фреймворк, який виконував би Javascript код на стороні сервера. Він розумів, що це може підвищити продуктивність. Його ідея створити фреймворк для створення серверів на Javascript з асинхронними вводом і виводом спрацювала на відмінно. Node.js, що став не просто фреймворком але середовищем, платформою для виконання Javascript коду, допоміг вирішити проблему C10M Problem. Яким чином? Node.js використовує орієнтовану на події модель та неблокуючу архітектуру вводу-виводу. Головна магія, яка відбувається в Node.js – цикл подій (рис. 1.2). Це буквально нескінченний цикл і насправді один потік.

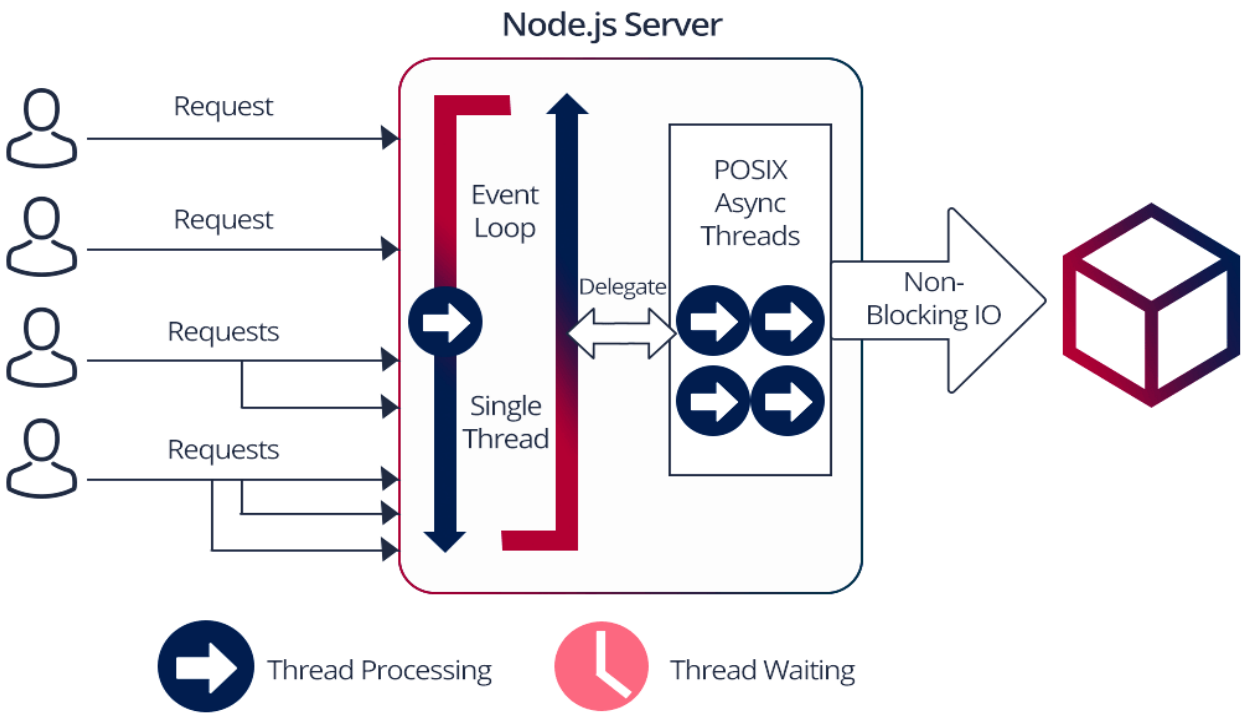

Рис.1.2. Схема Node.js серверу

Цикл має 6 фаз, кожне виконання фази називають tick:

1. Timers – у цій фазі виконуються колбеки, заплановані методами setTimeout() и setInterval().

2. Pending callbacks – виконуються майже всі колбеки, за виключенням подій close, таймерів и setImmediate().
3. Idle, prepare – використовується тільки для внутрішніх цілей.
4. Poll – відповідальний за отримання нових подій вводу-виводу. Node.js, насправді, може блокуватися на цьому етапі.
5. Check – колбеки, викликані методом setImmediate(), виконуються на цьому етапі.
6. Close callbacks, наприклад, socket.on('close', ...).

Є один потік, і він є циклом подій, але хто тоді виконує всі операції вводу-виводу? Коли циклу необхідно виконати операцію вводу-виводу він використовує потік ОС з тредпулу (thread pool), а коли задача виконана, колбек стає у чергу під час фази pending callbacks. За ці операції відповідає бібліотека Libuv, яка є частиною ядра Node.js. Але нічого не може бути ідеальним, і у Node.js є проблема CPU об'ємних задач – коли головний потік виконує важкі обчислення він стає недоступним для інших задач. Починаючи з версії v10.5 в Node.js присутній нативний модуль Worker Threads, який використовується для виконання CPU об'ємних Javascript операцій.

Аналізуючи інформацію з відкритих джерел, можна зробити деякі попередні висновки про Node.js. Платформа Node.js має наступні позитивні властивості:

- Власний інтерфейс на C++ для взаємодії з пристроями вводу-виводу комп'ютера. Тобто ця платформа фактично перетворює Javascript з вузькоспеціалізованої скриптової мови у мову загального призначення. І це означає, що Node.js підходить для побудови гібридних мобільних застосунків, комп'ютерних програм, ігор, вебсерверів, навіть для IoT (англ. Internet of Things – концепція мережі, що здійснює обмін даними між фізичними пристроями і комп'ютерними системами в автоматичному режимі, з використанням стандартних протоколів зв'язку).

- Застосунки на Node.js можуть виконуватися операційними системами Windows, Linux, MacOS та інших.
- Є full-stack ПЗ, адже дає можливість працювати як над клієнтськими (front-end), так і над серверними (back-end) частинами застосунків, використовуючи одну і ту ж саму мову програмування – Javascript.
- Відносна простота, швидкість та легкість написання программ.
- Можливість масштабування застосунків у всіх відношеннях. З Node.js легко обробляти тисячі синхронних запитів, потік даних відносно швидший і для обробки необхідно менше оперативної пам’яті.
- Node.js підходить для створення кроссплатформених застосунків. Такі технології дають змогу користувачу відправляти і отримувати повідомлення з телефону, веб версії застосунку та комп’ютерній програмі.
- Постійний розвиток завдяки рушію Google V8, що постійно розширює межі і є одним з найшвидших інтерпретаторів динамічних мов.
- Велика спільнота користувачів, що дає можливість отримувати вирішення проблемних питань, багато з них вже вирішені.
- Node package manager або npm – величезна кількість бібліотек які можуть бути встановлені написанням однієї строки в терміналі.

Також можна визначити очевидні мінуси використання Node.js:

- Швидкий розвиток призводить до необхідності постійно контролювати оновлення, щоб раніше розроблені застосунки продовжували працювати коректно. Наприклад, в минулому, один розробник видалив свою бібліотеку з npm і велика кількість застосунків, що користувалися бібліотекою перестали працювати.
- Підтримка об’ємних процесорних обчислень низька. Важкий запит на обчислення блокує обробку усіх інших завдань і уповільнить додаток, написаний на Node. Тому він не підходить для проєктів, що вимагають складних багатопоточних обчислень. Worker threads не можуть

забезпечити координованих обчислень між потоками і потребують більше пам’яті, ніж звичайна багатопоточна програма.

- Node.js вимагає чіткої архітектури, ця платформа орієнтована на події, тому вона може запускати декілька подій одночасно, але тільки якщо стосунки між ними добре прописані.
- Документація Node.js недорозвинена, у ній занадто багато альтернатив, тому не завжди зрозуміло, що вибрати.

Ми розглянули причини виникнення платформи Node.js як фреймворка Javascript, принципи її побудови, сильні та слабкі сторони. Можна зазначити, що Node.js перевершив сподівання його автора Райана Дала. І хоча основна мета його не змінилась – це простий спосіб побудови мережевих серверів які легко масштабуються та спонукання full-stack розробки на Javascript, але майбутнє цієї платформи, схоже, полягає у використанні її як обгортки, а наповнення писатиметься іншими мовами програмування.

1.3. Фреймворки, як засіб розробки вебзастосунків та інтерфейсів на платформі Node.js

Загалом, мови програмування і фреймворки використовуються для досягнення конкретних цілей проєкту, що є основними критеріями вибору. Для вирішення питання вибору фреймворків у проєктах розроблених на основі Node.js, ми зробимо огляд та аналіз основних Javascript фреймворків.

Програмний каркас або фреймворк (англ. software framework) – це готовий до використання комплекс програмних рішень, включаючи дизайн, логіку та базову функціональність системи або підсистеми. Фреймворк, як правило, містить тільки базові програмні модулі, а всі специфічні для проєкту компоненти реалізуються розробником на їх основі. На відміну від бібліотек, фреймворки – це компоненти архітектури програмного забезпечення. Програмна бібліотека може бути використана як набір підсистем з близькою функціональністю, без впливу на архітектуру основного програмного продукту і без накладання на неї

якихось обмежень. Тоді як фреймворк не просто дає розробнику потрібний функціонал, а також диктує правила побудови архітектури застосунку, завдаючи на початковому етапі розробки поведінку за замовчуванням, формуючи каркас, який можливо буде розширювати і змінювати відповідно до запланованих вимог. Фреймворк також може мати в собі допоміжні програми, бібліотеки коду, мови сценаріїв та інше програмне забезпечення, яке полегшує розробку і об'єднання різних компонентів великого програмного проєкту. Вони допомагають підвищити ефективність роботи та дозволяють розробнику звернути увагу на основну логіку програми. Завдяки цим інструментам автоматизуються деякі дії, які необхідно повторювати раз за разом. Розробка з використанням фреймворку позбавляє від проблеми використання повторюваного коду. Більшість із них використовують спільні бібліотеки та елементи які використовуються багаторазово та скорочують кількість операцій. Уніфікація структури для побудованих на основі фреймворку застосунках – це одна з головних переваг. Такі застосунки значно простіше супроводжувати і доопрацьовувати, адже стандартизована структура організації компонентів полегшує підтримку коду іншими програмістами, які в подальшому працюватимуть з ними. Розробка на основі фреймворку, на відміну від самописних рішень, дозволяє отримати спрощення у подальшому супроводі проєкту. Питання подальшої підтримки проєктів дуже важливе. Ось, як з цього приводу висловився Брайан Керниган – співавтор знаменитого довідника «Мова програмування С», мов AWK, AMPL і Ratfor, – «Налагодження коду вдвічі складніше за його написання. Тож, якщо ви пишете код настільки розумно, наскільки можете, то ви від початку недостатньо кмітливі, аби його налагоджувати».

У методології фреймворків зазвичай закладені найкращі практики програмної інженерії і дотримуючись цих правил можна уникнути багатьох проблем і помилок у проєктуванні. По суті фреймворк – це безліч конкретних і абстрактних класів, пов'язаних між собою і впорядкованих згідно з методологією фреймворка. Конкретні класи зазвичай реалізують взаємні відносини між класами, а абстрактні класи являють собою точки розширення, в яких закладений

у фреймворк базовий функціонал може бути використаний «як є» або адаптований до задачі конкретного додатка. Для забезпечення розширення можливостей в більшості фреймворків використовуються техніки об'єктно-орієнтованого програмування, наприклад, частини програми можуть успадковуватися від базових класів фреймворка або окремі модулі можуть бути підключені як домішки.

Більшість фреймворків для розробки вебдодатків використовує парадигму MVC. Тобто в багатьох фреймворках ідентичний підхід до організації компонентів програми і це спрощує розуміння архітектури додатку побудованого з використанням незнайомого розробнику фреймворка. MVC – шаблон архітектури, що використовується під час проєктування та розробки ПЗ. Проєктування з його використанням дозволяє поділяти системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс) та модуль керування (контроллер). Завдяки використанню MVC модифікація будь-якого з цих компонентів мінімально впливає на інші. Основна мета застосування MVC полягає в розділенні даних і бізнес-логіки та візуалізації. За рахунок чого збільшується можливість повторного використання програмного коду: наприклад додавання представлення даних якогось існуючого маршруту не тількиу вигляді HTML, але й, наприклад, у форматах JSON, PDF, таблицях Excel можна зробити дуже зручно, без потреби змінювання слоя бізнес-логіки початкового маршруту. Розглянемо компоненти MVC:

- Модель / Model – це об'єктна модель деякої предметної області, об’єднує у собі дані та методи роботи з ними, реагує на запити із контролера, повертаючі дані та/або змінює свій стан. При цьому модель не містить у собі інформацію про способи візуалізації даних чии форматах їх представлення, а також не взаємодіє з користувачем напряму.
- Представлення / View – відповідає за відображення інформації (візуалізацію). Ті самі дані можуть представлятися різними способами у різних форматах. Так колекцію об’єктів за допомогою різних представлень можна представити на рівні користувацького інтерфейсу

як у вигляді таблиці, так і списком; на рівні API можна експортувати дані в JSON чи таблицях Excel.

- Контроллер / Controller – логіка взаємодії користувача з системою. Використовує модель та представлення для реалізації необхідної реакції на дії користувача. Як правило, на рівні контроллера здійснюється фільтрація отриманих даних та авторизація – перевіряються права користувача на виконання дій або отримання інформації.
- Система маршрутизації – як додатковий компонент співставляє запроси з маршрутами та вибирає для обработки запитів певний контроллер.

В узагальненому випадку, коли до застосунку приходить запит, система маршрутизації вибирає потрібний контроллер для обробки запиту. Контроллер обробляє запит. В процесі обробки він може звертатися до даних через моделі і для рендерингу відповіді використовувати представлення. Результат обробки контроллера відправляється у відповідь клієнту. Нерідко відповіддю є html-сторінка, яку користувач бачить у своєму браузері.

Контролер забезпечує зв’язок між користувачем та системою. Він вирішує, які методи мають виконуватися у випадку тієї чи іншої дії користувача. Для реалізації необхідної дії він використовує модель та представлення. MVC є загальним шаблоном і не має строгої реалізації. Немає загальноприйнятого визначеного місця, де повинна розміщуватися бізнес-логіка додатку. Вона може знаходитись як у контролері, так і в моделі. Деякі фреймворки чітко задають, де має знаходитись логіка програми, інші не мають таких установок.

Для того щоб скористатися всіма можливостями фреймворків потрібен чималий багаж знань в розробці застосунків. Кожна людина має різні вподобання і потреби. Для одного розробника використання фреймворків може допомогти у прискоренні процесу програмування, а для іншого це може здатися марною тратою часу. У більшості випадків це залежить від рівня професіоналізму, але, загалом, фреймворки призначені, щоб заощадити час і абстрагуватися від рутинних завдань.

1.4. Висновки до першого розділу

Ми вказали на об’єктивні причини виникнення платформи Node.js, як фреймворка Javascript, принципи її побудови, сильні та слабкі сторони. Потрібно зазначити, що, виникнення таких технологій зробило популярнішим запит на full-stack фахівців, здатних реалізовувати всі частини проєкту. Можемо зазначити, що full-stack розробник – це спеціаліст, що бере активну участь на всіх етапах розробки проєкту. Він працює над створенням правильної структури, яка надалі зможе легко розростатись та витримувати максимальні навантаження. Full-stack розробник у фронтенді має такі основні задачі: робота з фреймворками, взаємодія з API та розгортання клієнтської частини на вебсервісах, можлива необхідність роботи з побудовою інтерфейсу і стилізуванням. У бекенді виникають наступні задачі: створення якісного API для фронтенду, розробка бізнес-логіки, інтеграція коду зі сторонніми API, робота з базою даних, розгортання продукту в інтернет (англ. deployment) можлива необхідність написання автоматичних тестів, робота з інструментами AWS, Azure, Google тощо.

Основним недоліком на шляху становлення full-stack фахівцем стає високий поріг входу через значну витрату часу на навчання. Об’єднання розробки серверної та клієнтської частин на основі мови програмування Javascript та використанням платформи Node.js, безумовно, дозволяє скоротити такі витрати. Але постає проблема вибору технологій і фреймворків для застосування.

Екосистеми фреймворків багаті на готові реалізації багатьох функціональних можливостей. Розробникам при роботі над типовими завданнями не потрібно щось вигадувати, адже вони можуть скористатися вже створеною спільною реалізацією. Це не тільки скорочує витрати часу і грошей, але і дозволяє досягти більш високої стабільності рішення, адже компонент, що використовується і допрацьовується тисячами інших розробників зазвичай більш якісно реалізований і краще протестований на всіляких сценаріях, ніж рішення, яке

може в адекватні терміни розробити один розробник або навіть невелика команда. Можлива відносно проста реалізація будь-якої бізнес логіки, а не тільки тої, що була закладена в систему спочатку. На сьогоднішній день, зазвичай, розробка без використання платформи ведеться в двох випадках – проєкт простий і не вимагає подальшого розвитку або дуже навантажений і вимагає глибокої оптимізації (наприклад вебсервіси з десятками тисяч звернень в секунду). В інших випадках розробка на основі програмної платформи вважається швидшою і якіснішою. Проєкти на базі фреймворків легше масштабуються та модернізуються. З точки зору бізнесу – розробка за допомогою фреймворка часто дає економічно більш ефективний і більш якісний результат, ніж написання проєкту чистою мовою програмування без використання будь-яких платформ.

РОЗДІЛ 2
ЗБІР, СИСТЕМАТИЗАЦІЯ ТА ДОСЛІДЖЕННЯ ІНФОРМАЦІЇ ПРО СУЧАСНІ РІШЕННЯ РОЗРОБКИ ВЕБЗАСТОСУНКІВ

2.1. Загальні проблеми оцінювання та вибору Javascript фреймворків для веброзробки

Фреймворки використовуються в різних областях залежно від їх основних характеристик. Оскільки існує безліч різновидів фреймворків і бібліотек – це сильно ускладнює процес вибору фреймворку для роботи окремим розробником. Під час вибору фреймворку можуть виникнути певні труднощі, пов’язані з визначенням завдань, які він може виконувати, та його призначення. Якщо для створення сайту потрібно знайти зручний і простий в освоєнні фреймворк, то необхідно ретельно підійти до питання його вибору та зважити всі «за» і «проти». Адже неправильний вибір фреймворку потенційно може стати однією з основних причин невдачі проєкту. Виділимо основні критерії вибору фреймворку в таблиці 2.1.

Таблиця 2.1

Критерії раціональності використання фреймворків

Назва	Характеристика
Підтримка баз даних	Питання підтримки баз даних у фреймворку дуже важливе. Наприклад, частина фреймворків мають вбудований ODM/ORM шар, частина – ні. Залежно від бази даних, яка використовується для розробки проєкту доводиться вибирати той чи інший фреймворк.
Підтримка спільноти	При розробці застосунків, часто виникають моменти, коли розробник сам не може впоратись з вирішенням якоїсь проблеми, але, за наявності великої спільноти готових допомогти розробників, можна вирішити більшість проблем.

Продовження Таблиці 2.1

Документація	Деякі фреймворки не мають достатньої кількості документації або вона не є актуальною. Тому, перед вибором фреймворка, необхідно переконатися в тому, що документація актуальна і вчасно оновлюється, доповнюється, а інструкція із застосування проста для розуміння.
Швидкодія	Пропускна здатність системи – кількість операцій або опрацьованих запитів за одиницю часу на певному апаратному забезпеченні, або інша подібна вимірювана величина. Від швидкодії вебсистеми залежить: кількість користувачів, які одночасно можуть працювати з нею (наприклад, кількість одночасних відвідувачів сайту), сатисфакція користувачів від взаємодії (прийнятність часу відповіді сайту на запити відвідувача суттєво впливає на те, чи користувач продовжить відвідування цього сайту і стане замовником, що платить, чи залишить його і, можливо, перейде до конкурентів та скористається їх послугами), як наслідок двох попередніх факторів – отримуваний від користувачів прибуток сайту, як через рекламну модель монетизації, так і прямий – від замовлень відвідувачів, що стали клієнтами.
Коректність	Задача гарантування коректності програмної системи завжди була і залишається складною і трудомісткою. У випадку використання сторонніх компонентів для її побудови (зокрема, фреймворків) задача стає практично нерозв’язною у класичному смислі (повністю) ще і з тієї причини, що частина системи (ті ж самі компоненти та фреймворки) стає «чорною скринею», у яку немає способу «зазирнути» і стверджувати напевно будь-що відносно функціонування.

Продовження Таблиці 2.1

Поріг освоєння	Не всі фреймворки прості в освоєнні. Це дуже важливо враховувати при виборі, так як на освоєння одного фреймворка може не вистачити й року, а на освоєння іншого – вистачить всього тижня. У першому випадку зміни у технології можуть наступити раніше, ніж будуть якісно опановані попередні стабільні версії.
Швидкість розробки	Треба оцінювати в комплексі, тобто : Час, необхідний для вивчення технології. Програмування з використанням технології. Відслідковування та застосування змін у технології (включаючи повторні тестування) – залежатиме від частоти та масштабності таких змін. Необхідність зміни у команді, що працює над проєктом. В результаті такої комплексної оцінки може виявитись, що вибраний фреймворк не підходить, тому що вимагає значно більше часу на повний цикл розробки програмної системи.
Архітектура	Не захищають від помилок проєктування архітектури програмної системи. Код, структурований розробником самостійно (скажімо, при класичному ООП), потребуватиме реорганізації, згідно визначених фреймворком правил. Це може зробити код менш цілісним і організованим складніше порівняно з тим же ООП.
Шаблони для створення інтерфейсів	Наявність даної функції значно спрощує створення інтерфейсу сайту, а іноді коли потрібен простий сайт то може звести написання інтерфейсу до мінімуму.

Продовження Таблиці 2.1

Швидкість розвитку	Цей пункт так само дуже важливий, адже деякі фреймворки оновлюються раз на пару років, а деякі – раз на пару місяців. Це дозволяє уникнути використання старого, не доопрацьованого коду при розробці. Водночас з випуском нових версій можливі суттєві внесення коректив до частин розробленої системи, якщо нові версії не мають зворотної сумісності (тобто не підтримують старі API, що цілком можливо у сучасному світі IT)
Створення і перевірка форм	Наявність даного функціоналу фреймворка полегшує створення полів для вводу тексту, логіну чи паролю до мінімуму. Полегшує створення валідації форм.

Визначившись з критеріями для оцінювання фреймворків, та побіжно ознайомившись із документацією, досліджувач цієї теми неодмінно стає на “слизьку” стежку вивчення різного роду опитувань, статистичних доповідей, статей та відгуків розробників на різних вебресурсах. Проблематика такого роду досліджень полягає у тому, що можна одночасно отримати різні результати, користуючись різними джерелами. Можна описати деякі загальні критерії для вірного оцінювання таких результатів:

- В першу чергу потрібно визначитись із задачами проєкту, умовами які будуть супроводжувати роботу, побачити питання, які не видно з першого погляду. Парадигма програмування – це набір ідей і понять, припущень та обмежень, концепцій, принципів, постулатів, прийомів і технік програмування для вирішення задач. У світлі піднятої теми в даній роботі можна сказати, що ми пишемо швидше на API, ніж на мові, а кожна мова має навколо себе інфраструктуру: у вигляді API, бібліотек, фреймворків і методологій, і обрані платформи та модулі повинні бути максимально “заточені” під ваш проєкт (технічно можливо все, але не варто створювати

інтернет-магазин на рушії для блогів, або на рушії для сайтів створювати складний корпоративний портал).

- Правильне використання наявних ресурсів. Кожен розробник обмежений проєктом, незалежно від рівня кваліфікації, займаної посади та розміру фінансування. Це означає, що список використовуваних технологій зумовлений та статичний (не останню роль в цьому грає фінансування) і оновлення стеку технологій може бути не скоро, або й ніколи.
- Якщо є команда розробників – треба прислухатися до їхньої думки. Адже важливі не тілки можливості та переваги самої платформи, але і наявність у розробників досвіду роботи саме з таким стеком. Звичайно, тут важливі не особисті уподобання, а корисні аргументи з відсиланнями на технічні особливості платформи необхідні для проєкту, а також фактичний досвід розробників саме з визначеною платформою.
- Ми вказували вище на одну з переваг фреймворків, а саме – готові способи реалізації різних функціональних можливостей. Компоненти, що використовуються і доопрацьовуються тисячами інших розробників, зазвичай, якісно реалізовані і краще протестовані на всіляких сценаріях.

2.2. Аналіз та порівняння Javascript фреймворків з використанням відкритих джерел інформації

Проаналізувавши декілька опитувань та публікацій присвячених цьому питанню у вільному доступі, можемо зробити висновок, що показник використання фреймворку і задоволеністю ним важливий, але, за умови, що опитування репрезентативне і охоплює великий зріз спільноти розробників.

Впевнено можемо пропонувати до вивчення опитування проведені командою “State of Javascript”. Репрезентаційний рівень опитувань найбільший – це більш як 21000 респондентів з 153 країн. Рівень довіри до сайту не викликає сумнівів. Фреймворки JS у опитуваннях “State of Javascript” розбиті на три основні групи:

1. Front-end фреймворки.
2. Фреймворки для завантаження та управління даними.
3. Back-end фреймворки – Javascript на сервері.

Ми виокремили фреймворки, що за рейтингом найбільш відомі, реакція на роботу з якими найбільш позитивна. У першій групі найбільш відомими є React.js, Angular та Vue.js, а за рівнем задоволення лідують React.js, Svelte.js та Vue.js (рис. 2.1)

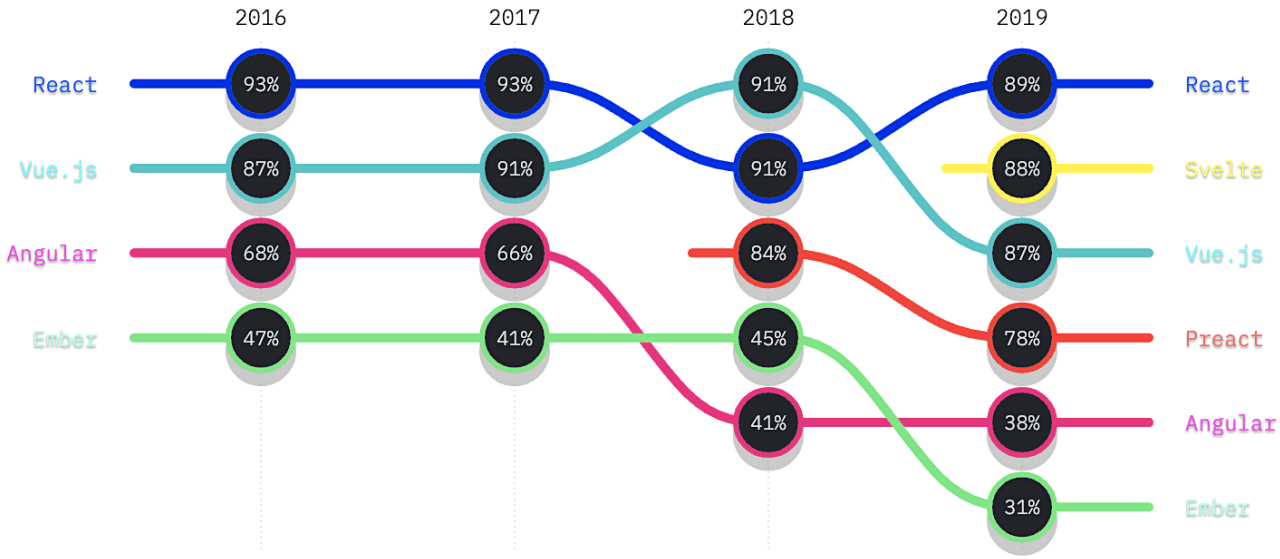

Рис. 2.1. Графік задоволення користувачів front-end фреймворків

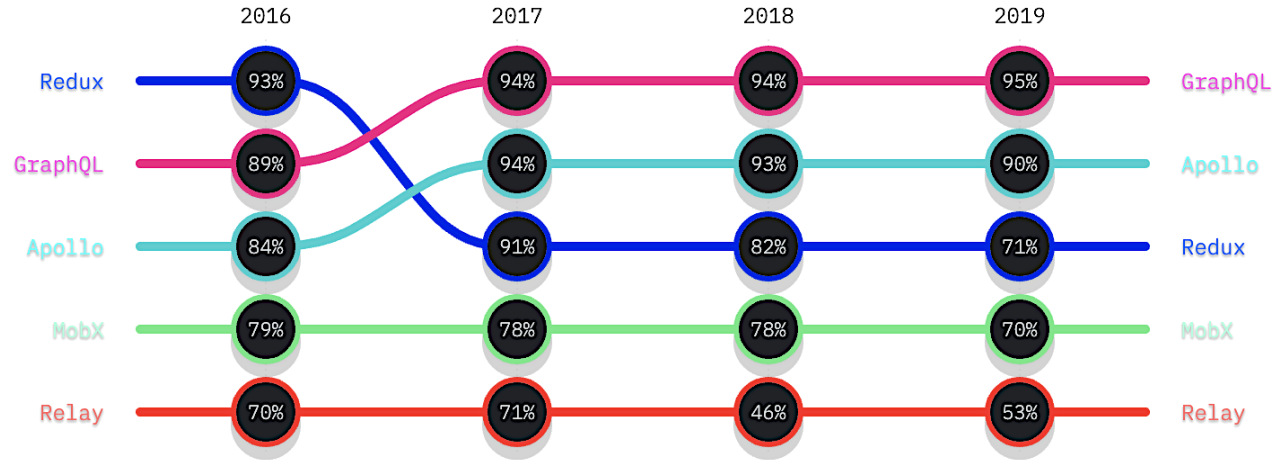

Рис. 2.2. Графік задоволення користувачів фреймворків для роботи з даними

У другій виділено : GraphQL, Apollo.js, Redux.js (рис. 2.2)

У третій це: Express.js, Next.js, Gatsby.js (рис. 2.3)

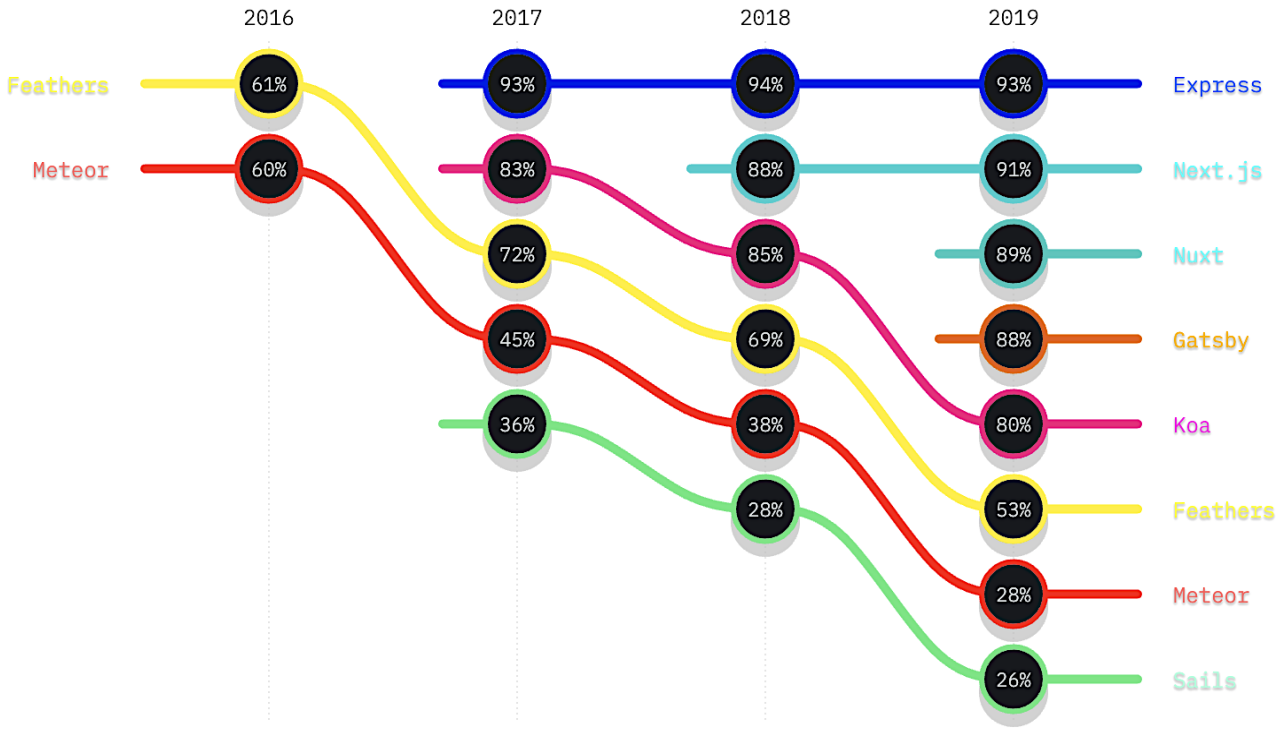

Рис. 2.3. Графік задоволення користувачів back-end фреймворків

React.js – open-source Javascript бібліотека для візуалізації (англ. rendering – візуалізація, проявлення, відмальовування, подання) бачення. React.js обробляє тільки користувацький інтерфейс у застосунках і все що з ним пов'язано. React дозволяє розробникам створювати великі вебзастосунки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. Розробник – Jordan Walke (Facebook, Instagram).

Angular – фреймворк для клієнтських застосунків, перш за все використовується для розробки SPA-рішень (Single Page Application), тобто односторінкових застосунків. Angular є наступником іншого фреймворку – Angular.js. Але це не нова версія Angular.js, у Google переписали Angular "з нуля", навіть ім'я змінили, повністю поламавши зворотну сумісність. Тому це принципово новий фреймворк. Розробник – Google.

Svelte.js – принципово новий підхід до створення інтерфейсів користувача. Традиційні фреймворки, такі як React.js та Vue, виконують основну частину своєї

роботи у браузері, Svelte переносить цю роботу на етап компіляції, який відбувається при збиранні застосунку. Svelte.js пише код, який точково оновлює DOM при зміненні стану вашого застосунку. Розробник – Rich Harris.

GraphQL – синтаксис, що описує як робити запити даних, переважно, використовується клієнтом для завантаження даних з серверу. Фактично, шар GraphQL знаходиться між клієнтом та одним чи кількома джерелами даних; він приймає запити клієнтів та повертає необхідні дані у відповідності до переданих інструкцій. Спочатку розробкою займалась Facebook до 2012, потім GraphQL Foundation увійшла до Linux Foundation.

Apollo.js – це платформа для побудови графа даних, комунікаційного рівня, який безперешкодно з'єднує ваші клієнти застосунків (наприклад, програми React та iOS) із вашими сервісними службами. Це реалізація GraphQL, яка допомагає вам переправляти дані з хмари до вашого UI. Ця платформа може бути накладена як додатковий шар на існуючі сервіси, включаючи REST API та бази даних. Розробник – Apollo.

Redux.js – бібліотека управління станом для застосунків, написаних на Javascript. Вона допомагає писати застосунки, які поводяться стабільно/передбачувано, працюють з різними оточеннями (клієнт/сервер/нативний код) і легко тестуються. В Redux.js стан застосунку представлений одним об’єктом Javascript – state (стан) або state tree (дерево станів). Незмінюване дерево станів доступне тільки для читання, змінювати напряму нічого не можна. Розробники – Ден Абрамов ,Марк Эріксон .

Next.js – фреймворк Javascript, заснований на React.js для створення SSR застосунків. Допомагає створювати призначений для користувачів інтерфейс застосунків (найчастіше, за допомогою React, не дотримуючись його принципу – SPA (Single Page Application). Він забезпечує загальну структуру, яка дозволяє легко створювати додаток React з зовнішнім інтерфейсом, і прозоро обробляє рендеринг на стороні сервера. SSR – Принцип, що використовується Next.js – рендеринг на стороні серверу. SSR допомагає знизити навантаження на пристрій, адже більшість операцій вироблених в додатку, відбувається на сервері, а не на

пристрої користувача. При відкритті сторінки, Next.js завантажує тільки той Javascript, який необхідний для цієї конкретної сторінки. Це гарантує, що завантаження першої сторінки буде настільки швидким, наскільки це можливо, і тільки майбутні завантаження сторінок (якщо вони коли-небудь будуть запущені) відправлять Javascript, необхідний клієнту. Розроблений Zeit у 2016, підтримується Vercel.

Express.js – найбільш скачуваний фреймворк для Node.js, називають «фреймворком для фреймворків» оскільки існує багато фреймворків, побудованих з його допомогою, надає широкий спектр утиліт HTTP. Хоча сам Express.js доволі мінімалістичний, розробники створили сумісні пакети проміжного програмного забезпечення для вирішення практично будь-якої проблеми з веброзробкою. Існують бібліотеки для роботи з кукі файлами, сеансами, автентифікація користувачів, параметрами URL, запитами POST, заголовками безпеки та іншими. (у 2015 розроблений TJ Holowaychuk з IBM (StrongLoop), зараз – Node.js Foundation).

Gatsby.js – це фреймворк, написаний на React.js. Даний фреймворк побудований на JAMStack архітектурі – це SSG, який дозволяє побудувати або перебудувати PWA сайт з дуже швидким завантаженням сторінок. Завантаження сторінок цілком і перехід між ними вимірюється в мілісекундах, а не секундах. Це приваблює як розробників, так і звичайних користувачів. Gatsby.JS може бути встановлений на ноутбук, генерувати статичні файли після зміни контенту з різних джерел і завантажувати їх на файловий сервер. Розробник – Kyle Mathews і Team Gatsby Inc.

Для формальної оцінки якості програмного забезпечення також використовуємо статистику з ресурсу github.com, що показує певні властивості програмного продукту або його специфікації у табл. 2.2.

Таблиця 2.2

Список фреймворків

Назва	**Кількість зірок на Github.com**	**“Використовував і буду надалі” на “State of Javascript”**
React.js	160 630	71,7% (у 2016 – 47,23%, чув, хочу вивчити – 12%)
Angular	59 588	21,9% (Не буду користуватися далі – 35,8%, чув, хочу вивчити – 9,7%, раніше – 39,7%)
Svelte.js	40 323	6,8% (чув, хочу вивчити – 44,9%)
GraphQL	17 000	38,7% (у 2016 – 5%, чув, хочу вивчити – 50,6%)
Apollo.js	17033	24,9% (у 2016 – 1,5%, чув, хочу вивчити – 36,1%)
Redux.js	54 900	47,7% (у 2016 – 36,5%, чув, хочу вивчити – 18,5%)
Next.js	58 500	24,7% (у 2016 – 8,6%, чув, хочу вивчити – 43,1%)
Express.js	51 200	71,6% (у 2016 – 61,3%, чув, хочу вивчити – 12%)
Gatsby.js	48 200	22% (чув, хочу вивчити – 35%)

Розглянувши деякі непрямі методи оцінювання, ми, безумовно, маємо приділити увагу загальноприйнятому способу перевірки програмного продукту – тестуванню. Тестування – це прикладна, стандартизована, інженерна практика. Тестування, як філософія, метрика або практика, існує набагато більше, ніж програмування. Під метрикою на увазі кількісний або якісний показник, що відображає успішність продукту або певну його характеристику. З її допомогою можна зрозуміти, наскільки актуальне для користувачів продукт або послуга компанії, чи допомагає він у вирішенні їхніх проблем і чи подобається. Для тесту продуктивності, оцінки якості і валідності коду використовують прикладне програмне забезпечення для аналізу коду.

Розробники міркують творчо. Тестувальники дивляться на вироблені розробниками речі деструктивно – вони з самого початку думають, як зламати запропонований розробниками продукт, адже розробники іноді слабо уявляють собі, що буде, якщо змінятся вхідні дані, якщо користувач буде активно повторювати якісь нетипові операції. Для прикладу ми взяли готові приклади перевірки коду. Для тесту продуктивності використані мініфіковані версії вихідних кодів фреймворків, в той час як для оцінки якості і валідності коду використані версії для розробки. Метрики були обчислені за допомогою прикладного програмного забезпечення для статичного аналізу коду Scitools Understand 4.0, утиліт plato і complexity-report. Результати підрахунку метрик обраних фреймворків представлені в табл. 2.3.

Таблиця 2.3

Результати розрахунку метрик вихідного коду фреймворків

метрика ПО	**Angular**	**Next.js**	**Ember**	**React.js**	**Vue**
Кількість рядків коду	28363	1155	25604	2127	6218
кількість виразів	14795	725	12680	873	2653
Кількість рядків коментарів	5	489	19171	1206	995
Співвідношення кількості рядків	~0	0,42	0,75	0,57	0,16
коментарів і рядків коду	3542	295	3432	414	1172
цикломатична складність	116	108	113	108	105
MI	9	7	10	8	8

З результатів значень метрик були отримані наступні висновки: Angular майже не містить коментарів, має високу вкладеність блоків – 9 і найвищу цикломатична складність (тобто кількість шляхів виконання програми) в своєму

коді, що сильно ускладнює його підтримку. Схожі результати по цикломатичній складності і глибині вкладеності блоків має Ember. У той час як Ember, Backbone і React містять в своєму коді більше 50% рядків коментарів, що говорить про високу документованості їх вихідного коду. Найнижчі показники цикломатичної складності і вкладеності блоків коду мають Next.js і React, що покращує їх підтримку і налагодження проєктів, заснованих на них. Тести валідації були проведені за допомогою утиліти Javascript Lint. У табл. 4 наведені помилки і попередження, знайдені при валідації розглянутих фреймворків табл. 2.4.

Таблиця 2.4

Результати валідації коду фреймворків

Назва фреймворка	**Помилки**	**Попередження**
Angular	1	19
Next.js	0	49
Ember	0	1
React.js	1	101
Vue	1	175

З результатів валідації були отримані наступні висновки. Angular, React.js і Vue містять по одній помилку в їх вихідному коді, що може призвести до не коректного поводження заснованих на них додатків. React.js і Vue містити найбільшу кількість попереджень, проте вони не є критичними. Ember має найвищі показники валідації коду.

Для проведення тестування продуктивності розглянутих фреймворків необхідно визначитися з предметом оцінки. Найбільш коректним буде використання вебдодатків, що мають абсолютно ідентичну функціональність, але реалізованих за допомогою різних фреймворків і мінімальною кількістю сторонніх бібліотек – лише тих, без яких практичне застосування фреймворків

неможливо. Під ці критерії підходять застосунки список справ (англ. to do list). Основним інструментом вимірювань є інструменти розробника браузера Chrome. Результати тестування продуктивності представлені в табл. 2.5.

Таблиця 2.5

Швидкість завантаження Todo застосунків.

Назва фреймворка	Angular	Next.js	Ember	React.js	Vue
Середній час завантаження Todo вебзастосунків, мс	639,85	317,97	898,35	470,09	265,4

З результатів проведення тестування продуктивності отримані наступні висновки: найкращі показники швидкості завантаження демонструють Vue, Next.js і React.js, в той час як Angular і Ember показують повільніші результати.

Для вибору підходящого фреймворку розробник повинен визначитися, який критерій буде мати більшу значимість в рамках потрібного проєкту на підставі технічних вимог. Суттєвим обмеженням проведеного порівняльного аналізу є те, що він враховує лише формальні «академічні» критерії оцінки програмного забезпечення. У той час як на практиці вибір фреймворку в більшості випадків ґрунтується на його популярності, повноті документації, частоті випуску оновлень, архітектурних перевагах.

2.3. Аналіз фронтенд-фреймворків

Фронтенд-фреймворки (фреймворки на клієнтській стороні) є невід'ємною частиною розробки на платформі Node.js. Angular, у порівнянні з двома іншими фреймворками React.js та Svelte.js, більш зрілий, і довкола Angular існує велика спільнота. Окрім того, що Angular є частиною знаменитого стека MEAN, він предоставляє деякі чудові функції, такі як двостороння прив’язка даних, впровадження залежностей, архитектура MVC, Angular CLI, підтримка

TypeScript, директиви и т. і. Але з появою конкурентів, таких как React.js , Vue, Svelte.js – Angular втратив популярність за останній час. Оскільки він є важким фреймворком , не виправдовуючи сподівань користувачів на оновлення, обмежена підтримка SEO та труднощі з вивченням, вплинули на популярність Angular досить різко. Та Angular все таки використовується для підтримки багатьох популярних вебсайтів і вебзастосунків, таких как Guardian, Upwork, PayPal та Sony. На цих великих сайтах Angular добре себе зарекомендував.

Ми дійшли висновку, що Angular найбільш використовуваний у наступних випадках:

- Великі масштабні програми.
- Необхідність масштабування архітектури.
- Використання TypeScript.
- Створення застосунків реального часу.

Ми зрозуміли, що порівняння між цими трьома технологіями не може бути повністю об’єктивним, або точніше сказати коректним. Angular постачається з повною структурою (MVC), це монолітна конструкція, яка постачається з усім необхідним “з коробки“, тоді як React.js – це frontend-бібліотека з безліччю пакетів відкритим кодом. React.js постачається не з усім інструментарієм, але має безліч хороших варіантів для інтеграції. React.js є більш гнучким у порівнянні з Angular, оскільки розробникам доведеться працювати з незалежними бібліотеками із порівняно кращим часом відгуку. Ще дійсно чудово в React.js з точки зору продуктивності – це Virtual DOM. Для кожного об’єкта DOM існує представлення (копія) цього об’єкта DOM. React використовується в основному для V (перегляду) в моделі MVC, оскільки інтерфейс користувача можна оновити не звертаючись до сервера та отримати новий вид. React.js є декларативний та компонентний, вебсторінки розділені на невеликі компоненти для створення інтерфейсів користувача. React.js заснований на компонентах багаторазового використання. Простіше кажучи, це блоки коду, які можна класифікувати як класи або функції. Кожен компонент представляє певну частину сторінки, таку як логотип, кнопка або поле введення. Використовувані

ними параметри називаються реквізитами, що означає властивості. Компонентна функція корисна, коли мова йде про підтримку коду під час роботи з масштабними проєктами. Компоненти React.js зазвичай написані на JSX, але також є можливість використання звичайного Javascript. JSX – це надбудова над Javascript, яка дозволяє використовувати XML-подібний синтаксис в Javascript. Код написаний на JSX компілюється у виклики методів бібліотеки React.js. Браузер не може розуміти Javascript файли, які містять JSX код. Спочатку вони повинні бути трансформовані в звичайний і зрозумілий Javascript. Це відбувається в процесі транспіляції.

Найпопулярнішим способом транспіляції є використання Babel, що є дефолтною опцією при запуску `create-react-app`, так що, якщо її використовувати, то не варто турбуватися, там все відбудеться саме собою.

React.js можна використовувати на стороні сервера а також на стороні клієнта, що дозволяє розподілити навантаження візуалізації від сервера до клієнта, якщо це необхідно (рис. 2.4).

Також існує React Native. За назвою може здаватися, що це той самий інструмент, але це не так. React Native потрібен для створення мобільних додатків, що створюються під обидві популярних платформи (IOS і Android) одночасно. Обидві версії програми будуть відповідати рекомендаціям Apple і Google, при цьому розробка відбувається швидше і задіє одну і ту ж команду розробників. Тому розробка додатків на React Native дозволяє досягти максимальних результатів в найкоротші терміни.

Вебфреймворки, такі як Angular, React.js і Vue.js ґрунтуються на принципі очікування завантаження коду для збірки віртуального DOM. Тільки після цього вони можуть малювати сторінку, використовуючи бібліотеку. Svelte.js відрізняється від цих найпопулярніших фреймворків бо не використовує віртуальний DOM. Svelte.js був задуманий як фреймворк, але, по суті, це компонентний фреймворк, створений для компіляції компонентів на етапі складання, що дозволяє загрузити на сторінку лише необхідне для відображення

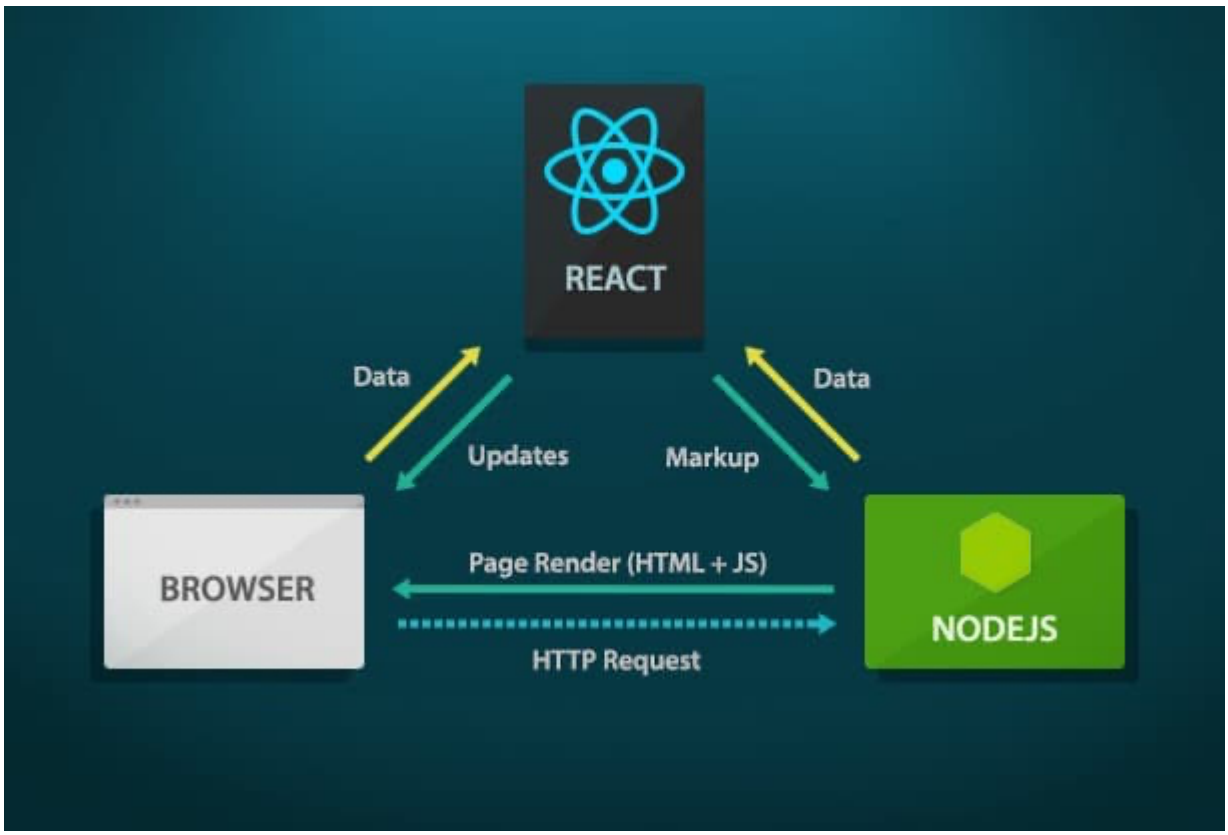

Рис. 2.4. Схема роботи React.js

вашого застосунку. Завдяки такому підходу, можна завантажувати лише один bundle.js на сторінку для рендера всього програми. Простіше кажучи, за допомогою Svelte.js ви пишете компоненти, використовуючи HTML, CSS і Javascript. В процесі розробки фреймворк компілює їх в невеликі автономні Javascript-модулі. Таким чином гарантується виконання мінімуму роботи з боку браузера, що робить веб-додаток швидше, а написання коду – простіше.

Зменшення кількості коду, який потрібно написати, є явною метою Svelte.js. Для ілюстрації давайте розглянемо дуже простий компонент, реалізований у React.js, Vue та Svelte.js. По-перше, версія Svelte.js:

```
<script>
 let a = 1;
 let b = 2;
</script>
<input type="number" bind:value={a}>
<input type="number" bind:value={b}>
<p>{a} + {b} = {a + b}</p>
```

Результат: 1 + 2 = 3 (рис. 2.5)

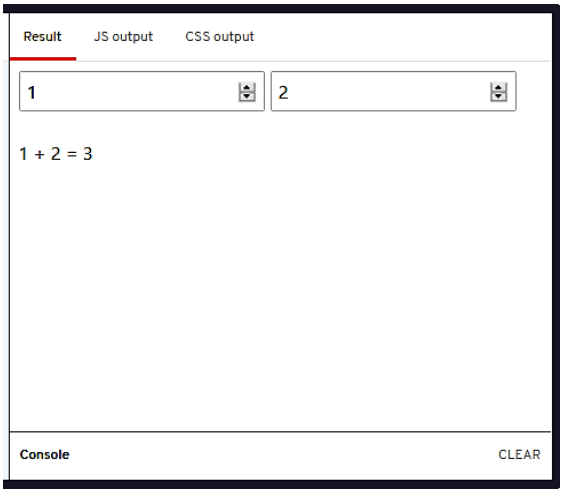

Рис. 2.5. Калькулятор на Svelte

Всі зміни введених даних одразу після вводу відображають оновлену суму у полі відповіді. Скомпільований Javascript код цієї програми має 2300+ символів. Такий самий застосунок, написаний на React.js:

```
import React, { useState } from 'react';
export default () => {
      const [a, setA] = useState(1);
 const [b, setB] = useState(2);
 function handleChangeA(event) {
       setA(+event.target.value);
 }
 function handleChangeB(event) {
       setB(+event.target.value);}
return (
  <div>
   <input type="number" value={a} onChange={handleChangeA}/>
   <input type="number" value={b} onChange={handleChangeB}/>
<p>{a} + {b} = {a + b}</p>
  </div>
 );
};
```

Іншими словами, для реалізації чогось, що займає 145 символів у Svelte.js, потрібно 442 символи в React .js та 263 символи у Vue. Версія React.js буквально втричі більша. Ще одна приємна річ – компоненти Svelte.js імпортуються так само легко, як і традиційні компоненти.

```
<div>
 <Wizardry />
```

```
</div>
<script>
 import Wizardry from './wizardry.html';
</script>
```

Порівняємо розміри пакетів деяких найкращих фреймворків, та їх розміри, включаючи стиснуті. Нижче наведемо табл. 2.6, в якій зібрані результати. Всі розміри в таблиці перевіряли на Bundlephobia

Таблиця 2.6

Розміри пакетів фреймворків

Framework	Bundle Size	Compressed
Angular	180.3kB	62.2kB
Vue	63.7kB	22.9kB
Preact	10.4kB	4kB
React	6.3kB	2.6kB
Svelte	3.8kB	1.5kB

Svelte.js значно менший за більшість фреймворків, оскільки він не має залежностей. Єдиними залежностями, які має Svelte.js – це залежності розробника, і це вражає. Що стосується Svelte.js та його екосистеми – не дивлячись на простоту та мінімалізм, Svelte.js досить таки повнофункціональний. Цей фреймворк має абсолютно все, що необхідно для створення сучасного UI. Svelte.js дозволяє писати високорівневий декларативний код у первинному коді, який просто розвивати та підтримувати, а на виході отримувати низькорівневий імперативний, високоефективний код, який відмінно працює у фазі виконання. Ті, хто порівнюють Svelte.js з іншими фреймворками, не завжди розуміють, що Svelte.js може запропонувати більш витончені рішення тих самих задач.

2.4. Аналіз фреймворків для роботи з даними

Вебсайти використовують бази даних для зберігання інформації як для користувачів, так і про користувачів. Фреймворки часто надають масив бази даних, який абстрагує операції зчитування, запису, запиту та видалення бази даних. Цей рівень абстракції називається Object-Relational Mapper (ORM).

Дані бувають закодовані в HTTP-запиті по різному. Для отримання файлів або даних з серверу HTTP-запит GET може кодувати, які дані вимагаються в URL-параметрах або у структурі URL. HTTP-запит POST для оновлення ресурсу на сервері замість цього буде включати оновлену інформацію як «POST дані» всередині тіла запиту. HTTP-запит може також включати інформацію про поточну сесію або про користувача в cookie на стороні клієнта. Фреймворки надають у відповідністю з програмною мовою механізми доступу до цієї інформації. Використання ORM надає дві переваги:

- Можна замінити базу даних, що лежить в основі без необхідності змінювати код, який її використовує. Це дозволяє розробникам оптимізувати характеристики різних баз даних в залежності їх використання.
- Може бути реалізована базова перевірка даних. Це дозволяє легше та безпечніше перевірити, що дані зберігаються у правильному полі типу бази даних, і мають правильний формат (наприклад адресу електронної пошти) та не є шкідливими (зломщики можуть використовувати певні шаблони коду, щоби зробити такі речі, як видалення записів бази даних).

Вебфреймворки часто надають системи шаблонів. Вони дозволяють вказувати структуру вихідного документу, використовуючи заповнювачі для даних, які будуть додані при створенні сторінки. Шаблони часто використовуються для створення HTML, але можуть також створювати інші типи документів. Також вони часто надають механізм, який дозволяє легко створювати інші формати з даних, що зберігаються, включаючи JSON.

Одним із лідерів опитувань є GraphQL, який можна описати як «мова запитів для API та середовище для виконання на стороні сервера з використанням системи типів, яку розробник визначає для своїх даних». Він не прив’язаний до якоїсь бази даних чи сховищу; за ним стоять код розробника та дані. У двох словах, GraphQL це синтаксис, який описує як зробити запит даних, він в основному, використовується клієнтом для завантаження даних з сервера. Можна виділити три основні характеристики у GraphQL:

- Дозволяє клієнту точно вказувати, які дані йому потрібні.
- Полегшує приєднання даних з декількох джерел.
- Використовує систему типів для опису даних.

Те, що він був створений саме в Facebook для проєкту з великим об'ємом різних даних, говорить про те, що при роботі над подібним проєктом можуть виникати ситуації з обмеженнями REST-архітектури. Наприклад, отримання профілю користувача, його постів і коментарів з самого початку не здається складним завданням. Але якщо врахувати обсяг даних в системі і припустити, що всі ці дані зберігаються в різних базах даних (наприклад MySQL і MongoDB), стає зрозуміло, що для цього потрібно буде зробити кілька REST-endpoint-ів. Ну а якщо уявити, наскільки великий обсяг даних і різнорідні джерела даних, то стає зрозуміло, чому потрібно було розробляти новий підхід для роботи з API. В основі цього підходу лежить наступний принцип: краще мати один «розумний» endpoint, який буде здатний працювати зі складними запитами і повертати дані саме в тій формі і в тому обсязі, які необхідні клієнту.

Якщо порівнювати GraphQL та REST то можна виділити у цих 2 підходів спільні риси:

- Вони надають єдиний інтерфейс для роботи з віддаленими даними.
- Найчастіше на запит повертається відповідь в форматі JSON.
- Обидва підходи дозволяють диференціювати запити на читання і на запис.

Але у GraphQL є і багато переваг у порівнянні з REST:

- Він клієнтоорієнтований, тобто дозволяє клієнту отримати ту інформацію і саме в тому обсязі, які йому потрібні – не більше і не менше.
- На відміну від REST, необхідний тільки один endpoint для роботи.
- GraphQL – сильно типізований мову, що дозволяє заздалегідь оцінити правильність запиту до етапу виконання програми.
- GraphQL надає можливість комбінувати запити.
- Запити до GraphQL API завжди повертають очікуваний результат, який відповідає схемі даних цього GraphQL API.
- Використання GraphQL дозволяє зменшити кількість даних, переданих клієнтові, так як клієнт повинен вимагати лише необхідні йому дані. Тобто якщо при використанні REST кінцева точка визначає те, які дані і в якій формі отримає клієнт, то при використанні GraphQL клієнт не повинен запитувати зайві дані, тим самим зменшуючи обсяг відповіді сервера.

На практиці використання GraphQL збільшує незалежність front-end і back-end-розробників. Після узгодження схеми front-end-розробники більше не будуть просити створити нові endpoint-и або додати нові параметри для наявних: back-end-розробники один раз описують схему даних, а front-end-фахівець створює запити та комбінує дані так, як це необхідно. Щоб почати роботу з GraphQL потрібні всього дві речі:

- Сервер GraphQL для обробки запитів до API.
- Клієнт GraphQL, який буде підключатися до endpoint.

Для того, щоб виконати GraphQL-запит, потрібен GraphQL-сервер з яким можна такий запит відправити. GraphQL-сервер являє собою звичайний HTTP-сервер (зазвичай це сервер, створений за допомогою Node.js та Express Framework), до якого приєднана GraphQL-схема. Задля користування GraphQL, необхідно встановити проміжне ПЗ apollo-server-express, що забезпечує пряме з'єднання між Express.js і сервером Apollo GraphQL. GraphQL і Apollo допомагають швидше завантажувати функції. У поєднанні цих платформ розгляд складних

питань, таких як кешування, нормалізація даних та оптимістична візуалізація інтерфейсу, стають простими.

Платформа Apollo.js дозволяє build, query, and manage a data graph : уніфікований шар даних , який дозволяє додаткам взаємодіяти з даними з будь-якої комбінації підключених сховищ даних і зовнішніх інтерфейсів.

Граф даних розміщується між клієнтами додатків та серверними службами, полегшуючи потік даних між ними (рис. 2.6). Граф даних Apollo.js використовує GraphQL для визначення та забезпечення структури цього потоку даних.

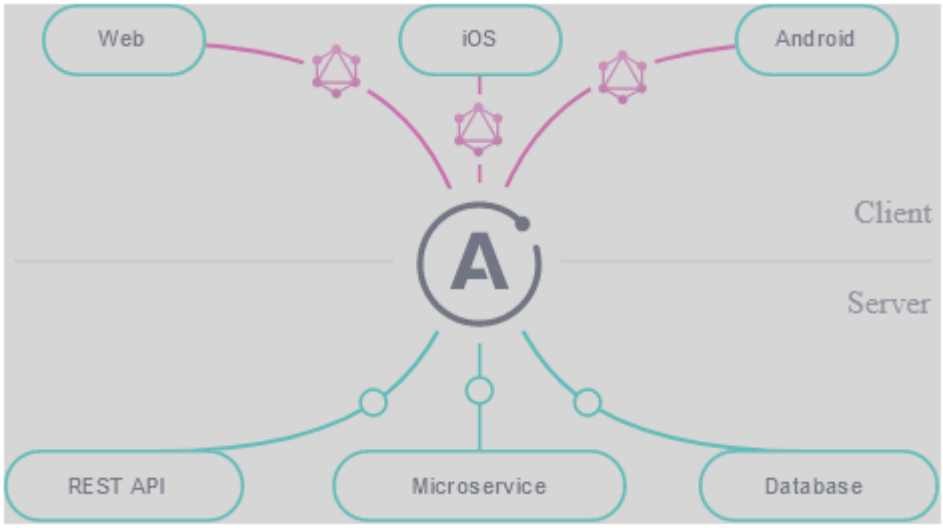

Рис. 2.6. Схема роботи Apollo

Графу даних потрібна послуга, яка обробляє операції GraphQL від клієнтів застосунків. Ця служба взаємодіє з внутрішніми джерелами даних для отримання та модифікації даних за потреби. Для створення цієї служби можна використовувати сервер Apollo .Apollo Server – це розширюваний сервер GraphQL з відкритим початковим кодом. За допомогою нього ви можете визначити:

- Схема GraphQL, яка визначає всі типи та поля, доступні у вашому графу.
- Колекція resolvers, що вказують, як заповнити кожне поле вашої схеми даними із ваших внутрішніх джерел даних.

Ви можете розгорнути Apollo Server у будь-якому розміщеному або безсерверному середовищі. Він підтримує різноманітні популярні програми NodeJS і безперебійно працює з TypeScript. На наведеній нижче схемі показана архітектура клієнт-сервер. Вебсервер побудований на Node.js і Express. Запит на сервер Apollo GraphQL виконується додатком ReactJS (створеним з використанням клієнтської бібліотеки Apollo) або браузерні додатком GraphQL (рис. 2.7). Запит буде проаналізовано і перевірено за схемою, визначеною на сервері. Якщо схема запиту проходить валідацію, то відповідні функції засобу розв'язання будуть виконані. Засіб розпізнавання буде містити код для отримання даних з API або бази даних.

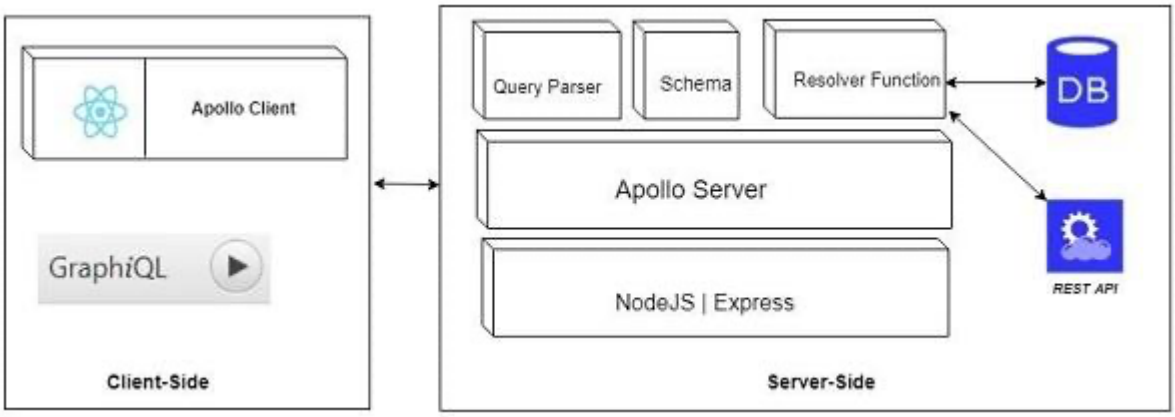

Рис. 2.7. Схема роботи Apollo

Ми бачимо, що деякі рішення на платформі Apollo.js з відкритим початковим кодом включають в себе:

- Apollo Server це сервер GraphQL, який працює з будь-якою схемою GraphQL, відповідної специфікації GraphQL.
- Apollo Client – це повнофункціональний клієнт GraphQL з підтримкою кешування з інтеграцією для React, Angular і інших інтерфейсних фреймворків. Він дозволяє легко створювати компоненти користувальницького інтерфейсу, які витягують дані за допомогою GraphQL.
- apollo-tooling – Apollo CLI об'єднує ваші клієнти і сервери GraphQL з інструментами для перевірки вашої схеми, перевірки ваших операцій на

сумісність з вашим сервером і генерації статичних типів для підвищення безпеки типів на стороні клієнта.

Типовий Apollo-client складається з двох частин:

- Apollo-client, що дозволяє виконувати запити GraphQL в браузері (також є розширення для DevTools).
- Конектор для frontend-фреймворка (React-Apollo, Angular-Apollo і інші).

За замовчуванням Apollo-client зберігає даних використовуючи Redux, який сам є досить авторитетною бібліотекою управління станом з багатою екосистемою.

Redux – це передбачуваний контейнер стану для Javascript додатків. Він вчить думати про програму, як про початковий стан, який змінюється послідовністю дій. Це допомагає писати програми, які поводяться послідовно, працюють в різних середовищах (клієнтських, серверних та власних) і легко тестуються. Крім того, це забезпечує чудовий досвід для розробників, наприклад, редагування коду в реальному часі. Можна використовувати Redux разом з React або з будь-якою іншою бібліотекою перегляду. Він крихітний (2 кБ, включаючи залежності), але має велику екосистему додатків. Redux Toolkit це офіційний рекомендований підхід до написання логіки Redux. Він охоплює ядро Redux і містить пакети та функції, які, на нашу думку, є важливими для створення програми Redux. Набір інструментів Redux використовує найкращі практики, спрощує більшість завдань Redux, запобігає типовим помилкам та полегшує написання програм Redux. Набір інструментів Redux доступний як пакет на NPM для використання з пакетом модулів або в програмі Node.js.

У Redux загальний стан застосунку представлено одним об'єктом Javascript – state (стан) або state tree (дерево станів). Дерево станів незмінне, доступне тільки для читання, напряму нічого змінити не можна. Зміни доступні тільки при відправці action (дії). Дія (action) – це Javascript-об'єкт, який лаконічно описує суть зміни: єдина вимога до об'єкта дії – це наявність властивості “type”, значенням якого зазвичай є рядок. При запуску дії обов'язково щось відбувається і стан додатки змінюється. Це робота редукторів. Редуктор -ред’юсер(reducer) –

це чиста функція, яка обчислює наступний стан дерева на підставі його попереднього стану і застосованої дії.

Чиста функція працює незалежно від стану програми і видає вихідне значення, приймаючи вхідне, не змінюючи нічого в ньому і, зрештою, в програмі. Виходить, що редуктор повертає абсолютно новий об'єкт дерева станів, яким замінюється попередній. В процесі розробки ред'юсери можуть бути розділені на дрібніші ред'юсери, які управляють певними частинами дерева станів. Оскільки ред'юсери – це лише функції, можна контролювати порядок їх надсилання, передавати додаткові дані або навіть створювати повторювані ред'юсери для звичайних завдань, таких як розбиття на сторінки. Потік даних в Redux завжди однонаправлений. Передача дій з потоками даних відбувається через виклик методу `dispatch ()` у сховище. Саме сховище передає дії редуктора і генерує наступний стан, а потім оновлює стан і повідомляє про це всіх слухачів. Redux зберігає стан всього застосунку в дереві об'єктів в одному сховищі. Одне дерево станів полегшує налагодження або перевірку програми; це також дозволяє зберігати стан вашого застосунку в процесі розробки, для прискорення циклу розробки.

Стор (Store) – це об'єкт, який з'єднує екшени, які представляють факт того, що "щось сталося" і reducer-и, які оновлюють стан (state) відповідно до цих екшенів разом.

- Стор містить стан додатку (application state).
- надає доступ до стану за допомогою `getState()`.
- може випускати оновлення стану за допомогою `dispatch` (action)
- обробляє скасування реєстрації слухачів за допомогою функції, що повертається `subscribe` (listener).

Redux ідеально підходить для середніх і великих застосунків. Ним варто користуватися тільки у випадках, коли неможливо управляти станом застосунку за допомогою стандартного менеджера станів в React або будь-який інший бібліотеці. Простим застосункам Redux не потрібен. Ось коли має сенс використовувати Redux:

- У вас є обґрунтовані обсяги даних, що міняються з часом.
- Вам потрібен один джерело інформації для вашого стану.
- Ви приходите до висновку, що зберігати все ваше стан в компоненті верхнього рівня вже недостатньо.
- Так, ці рекомендації є суб'єктивними і розпливчастими, але це справедливо. Точка, в якій ви повинні інтегрувати Redux в ваш додаток, відрізняється для кожного користувача і кожної програми.

2.5. Аналіз back-end фреймворків

Третя група досліджуваних у кваліфікаційній роботі – back-end фреймворки.

Express.js є найпопулярнішим вебсередовищем Node.js і є базовою бібліотекою для ряду інших популярних вебфреймворків Node.js. Він забезпечує наступні механізми:

1. Написання обробників запитів з різними дієсловами HTTP за різними шляхами URL (маршрутами).
2. Інтеграцію з механізмами візуалізації "view", щоб генерувати відповіді, вставляючи дані в шаблони.
3. Встановлення загальних налаштування вебзастосунків, такі як порт для підключення, розташування шаблонів, які використовуються для надання відповіді.
4. "Проміжне програмне забезпечення" для додаткову обробку запитів в будь-якій точці конвеєру обробки запитів у будь-який момент.

У той час як сам Express.js досить мінімалістичний, розробники створили сумісні пакети проміжного програмного забезпечення для вирішення практично будь-якої проблеми з веброзробкою. Існують бібліотеки для роботи з куки-файлами, сеансами, входами користувачів, параметрами URL, даними POST, заголовками безпеки і багатьма іншими. Ви можете знайти список пакетів проміжного програмного забезпечення, підтримуваних командою в Express Middleware

(поряд зі списком деяких популярних пакетів сторонніх виробників). Працюючі з Express.js можна використовувати базу даних, наприклад MongoDB з Mongoose (для моделювання), щоб забезпечити бекенд для застосування Node.js (рис. 2.8). Mongoose – це надбудова для драйвера Mongo , яка ідеально підходить для екосистеми Express. Головна перевага MongoDB полягає в тому, що він орієнтований на Javascript. Це означає, що драйвер передає всі дані, як вони є, не витрачаючи час на їх пристосування до середовища програмування. MongoDB розроблений для масштабованих серверів, керованих даними. Кожен запит в MongoDB обробляється так само, як і зворотний дзвінок . Це означає, що серверу не потрібно чекати (блокувати the stock), поки певні дані не будуть передані, і може одночасно обробляти інші запити. Це надзвичайно важливо, оскільки масштабована база даних може бути розташована на одному або декількох фізично віддалених серверах, тому отримання даних з них вимагає часу.

Це стало стандартною серверної платформою для node.js. Express.js - це внутрішня частина того, що відомо як MEAN-стек. MEAN - це безкоштовний програмний стек Javascript з відкритим вихідним кодом для створення динамічних веб-сайтів і веб-додатків, який має наступні компоненти: MongoDB, Express, Angular, Node.js. Інший варіант - MERN, із заміною Angular на React. Але на нашу думку, це насправді не стек, оскільки немає ОС. Хоча Node.js використовується не тільки для коду на стороні сервера, але й виконує функції веб-сервера, на практиці використовують Nginx як проксі-сервер і ставлять процес Node.js позаду, щоб краще обробляти трафік.

Ця схема (рис. 2.8) показує, як відбувається взаємодія таких елементів:

- Клієнти.
- Сервер API.
- База даних.

Клієнти можуть включати як вебплатформи, так і мобільні застосунки.

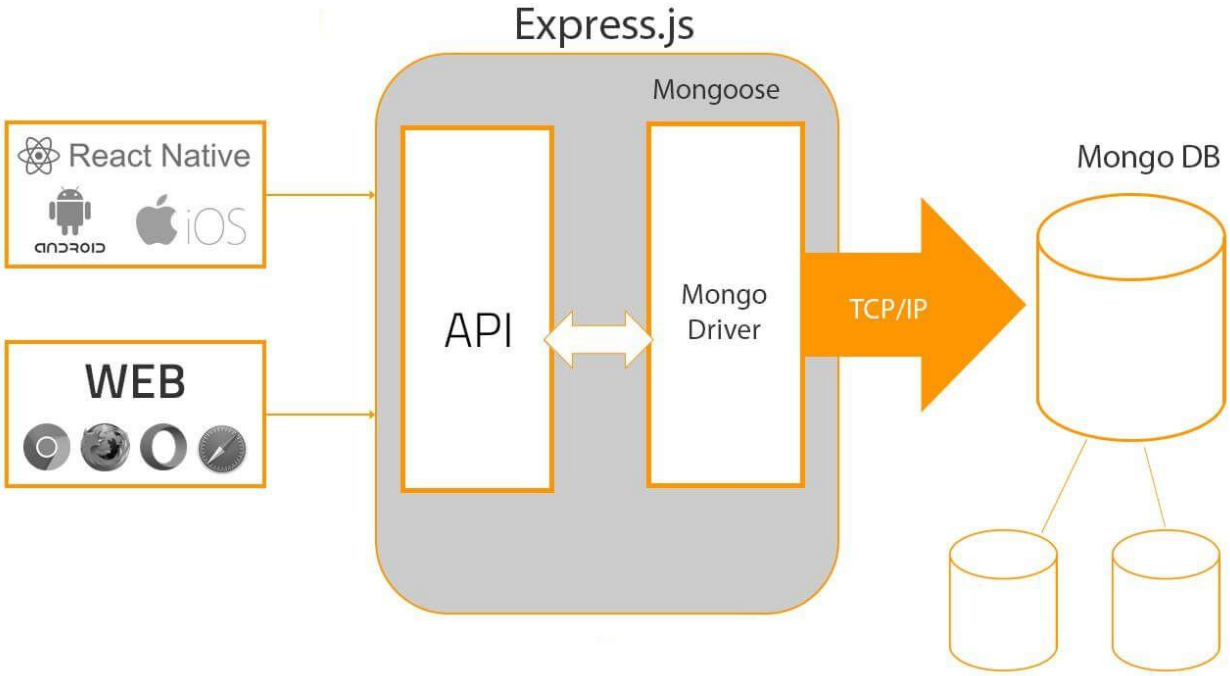

Рис. 2.8. Схема роботи Express

Вебплатформи можуть бути написані за допомогою будь-якої технології розробки (наприклад, React.js, Backbone.js). Мобільні програми можуть розроблятися на ReactNative, який підтримує платформи Android та iOS, або з рідною мовою програмування.

Сервер API обробляє всі запити від користувачів. Express.js ідеально підходить для систем з великим потоком входів і виходів. Всі дані обробляються та зберігаються у базі даних за допомогою Mongo Driver. Клієнти запитують дані із сервера через API . Сервер отримує дані з бази даних і повертає їх клієнту через API. Після цього клієнти відображають ці дані.

Важлива перевага це проста інтеграція сторонніх служб та проміжного програмного забезпечення. Express.js – це неперіодизована структура, що означає, що немає чітких і нечітких правил щодо того, як вирішувати певні завдання або які компоненти вибрати. Ви можете використовувати будь-яке проміжне програмне забезпечення у зручному для вас порядку. Крім того, ви можете вирішити, як структурувати програму, оскільки немає жодного правильного способу зробити це. Що дуже важливо – це одна з найбільш підтримуваних платформ Node.js. У ньому є спільнота з відкритим кодом, тому код завжди переглядається та вдосконалюється. Т. Дж. Головайчук, засновник

Express.js, заявив: “Я ніяк не міг навчитися всьому, що знаю, без відкритого коду. Правки та відгуки про ваш власний код дуже корисні. Звичайно, чудово, коли люди також насолоджуються чимось, що ви створили, так що це може бути весело".

Більшість розробників кажуть, що Express.js легко вивчити. Він добре задокументований та підтримується великою громадою. Більше того, той хто вже знайомий з Javascript та архітектурою з підтримкою, з легкістю почне з ним працювати. Визначимо деякі його недоліки:

- Побудований на філософії плагінів middleware, без розуміння суті яких працювати з фреймворком неможливо.
- Шаблонний код (при спробах змінити підхід в ході розробки будова коду буде порушено, що вимагає внесення додаткових правок).
- Node.js – це однопотокова структура з циклом подій, яка прослуховує різні події та виконує зареєстровані зворотні виклики-callbacks. Розробникам, які працювали з іншими мовами програмування, може бути важко зрозуміти природу callback-ів. Прочитавши кілька інструкцій, вони починають писати весь код у callbacks. Такий підхід лише заплутає код, тому його неможливо буде підтримувати в майбутньому. Це називається callbacks-пеклом. Остання версія Node.js дає можливість використовувати async / await, що підвищить якість коду та запобіжить проблемам із зворотними викликами.

Якщо створювати вишуканий вебсайт, що дозволяє власникам сайту, редакторам, авторам керувати їх сайтом та публікувати контент без жодних знань програмування, то в першу чергу згадують WordPress – найпопулярнішу систему управління контентом (Content Management System) в світі. Ця CMS лежить в основі понад 35% всіх сайтів в інтернеті, і це число постійно зростає. Але коли стратегія розвитку вимагає унікального функціоналу, важливі персоналізація контенту для користувачів і їх максимальне залучення у взаємодію з продуктом (на різних платформах), потрібно писати своє рішення і JavaScript буде більш перспективним. До того ж на ранжування сайтів впливає

швидкість завантаження. Сайти, зібрані на JavaScript, можуть цю швидкість забезпечити. Для забезпечення таких сайтів використовуються такі технології, як Next.js, Gatsby.js, Jekyll, Hugo, серед інших.

Gatsby.js – фреймворк, написаний на React.js. Це генератор статичних сайтів, за допомогою якого можна створювати повноцінні вебдодатки. Gatsby.js виділяється дуже високою швидкістю завантаження сторінок. Фреймворк використовує попередню завантаження: коли користувач відкриває головну сторінку, браузер у фоновому режимі завантажує дані, необхідні для відтворення інших сторінок сайту, на які є посилання з головної. Сайт на Gatsby.js є React-додаток, тому завантажуються тільки дані про різницю між сторінками, а не сторінка цілком. При переході оновлюється віртуальний DOM, відвідувач насолоджується високою швидкістю завантаження. Тож не дивно, що сьогодні 1% з 10 000 найкращих веб-сайтів побудовані за допомогою GatsbyJS, кілька назв ресурсів, що використовують його: PayPal, IBM, Braun, Ideo, Airbnb і Impossible Burger.

Gatsby.js створює повноцінні програми. Всі сайти, створені за допомогою Gatsby.js, підтримують функціональність PWA з коробки. Тобто власнику сайту нічого не треба робити – його ресурс вже є прогресивним вебдодатком з усіма перевагами цієї технології. Також Gatsby.js підтримує ефективну подгрузку сторінок і картинок і інші функції, які покращують perfomance сайту.

Найцікавіша можливість цього фреймворку представлена інструментом для роботи з GraphQL, який називається GraphQL. Gatsby використовує GraphQL. A GraphQL дозволяє працювати з даними, які використовуються на сайті. Треба підкреслити важливість і корисність цього інструменту. Він позбавляє розробника від необхідності читання документації по використовуваному джерела даних. GraphiQL дозволяє, в інтерактивному режимі, переглядати дані. З даних можна вибирати те, що потрібно. В результаті виходять автоматично створювані GraphQL-запити, які копіюють в компоненти. Недоліки цього фреймворку, які ми виділили:

- Немає справжньої системи серверного рендеринга.

- Інкрементні збірки і режим попереднього перегляду прив'язані до Gatsby Cloud.
- Схема кешування збірок Gatsby.js часто є причиною проблем з кешуванням.

Є ще один подібний React Framework – Next.js, який існує вже давно. Подібність між Gatsby.js та Next.js:

- Для розробника: Gatsby.js, та Next.js постачаються з великою документацією, яка допомагає розробляти функції над існуючим додатком. Вам не потрібно вчитися все з нуля, і базове розуміння React – це єдина обов’язкова умова обох фреймворків.
- Побудовані вебсайти: Gatsby.js та Next.js використовуються для створення високопродуктивних вебсайтів з дуже хорошими показниками Маяка, звичайно, якщо вони добре побудовані. Вебсайти зручні для SEO, оскільки обидва вони надають попередньо відтворений HTML.
- Корисні функції. Гаряче перезавантаження – як Gatsby.js, так і Next.js, постачаються з готовою функцією Hot Reloading. Інші цікаві вбудовані функції включають розділення коду, попереднє отримання, маршрутизацію та кешування.

Але подібність на цьому закінчується. Обидва фреймворки використовуються для принципово різних цілей. Gatsby.js ідеально підходить для створення статичних вебсайтів. Різні користувачі, які відвідують статичний вебсайт, бачитимуть однаковий вміст, а оновлення не відображатимуться в режимі реального часу. Особистий блог – хороший приклад статичного вебсайту.

Next.js – використовується для створення великих багатокористувацьких вебсайтів, із кількома користувачами, які створюють облікові записи користувачів, а потім коментують вміст на сторінці. На цих вебсайтах у будь-який момент приєднуються декілька користувачів, що унеможливлює створення статичного вебсайту під час роботи. На таких вебсайтах потрібно показувати унікальний динамічний вміст зареєстрованим користувачам. Візуалізація на

стороні сервера допоможе обслуговувати різних користувачів на основі аутентифікації, і кожен користувач може бачити їх вміст, щойно він створить його на вебсайті. Ключова різниця між Gatsby.js та Next.js – це спосіб обробки даних. Gatsby.js розділяє дані та вебсайт, це означає, що члени вашої команди, крім розробників, також можуть редагувати дані на вебсторінці, які потім можуть бути скомпільовані як сайт під час роботи. Next.js має підтримку Static Site Generation (SSG), що означає нові методи отримання даних.

Ці методи побудови часу називаються getStaticPaths та getStaticProps, і користувачі можуть використовувати їх для побудови статичних вебсайтів, як і у випадку з Gatsby.js:

- getStaticPaths – використовується для створення списку ідентифікаторів, які підтримує статичний вебсайт.
- getStaticProps – використовується для отримання даних, що генеруються на основі ідентифікаторів.

Використовуючи цю інформацію, Next.js створить сторінку для кожного з цих ідентифікаторів дописів у блозі, передаючи дані, отримані із зовнішніх служб, компоненту. Оскільки він поєднує в собі найкраще як статичне створення вебсайтів, так і рендеринг на стороні сервера, Next.js можна використовувати для створення ідеального вебсайту для електронної комерції. Можна зберігати деякі частини вебсайту статичними, як-от сторінку контактів або інформацію про доставку, а також використовувати сервер для відображення сторінок, що містять динамічний вміст, як-от сторінку продукту. Є ще кілька речей, про які потрібно пам’ятати розробникам:

1. Безпека даних: у Gatsby.js з джерела беруться лише необхідні дані, що робить їх більш безпечними. Хоча можна стверджувати, що з Next.js у вас є CMS та API, які мають приватні функції, дані все ще є на сервері, залишаючи відкритим шанс на використання.
2. Інфраструктура: за допомогою Next.js вам потрібно інвестувати в налаштування серверів з базами даних, обслуговування тощо. Тоді як за

допомогою Gatsby.js ви можете попередньо відтворити їх у збірці та використовувати CDN для створення надшвидких вебсайтів.

3. Зображення: плагін, що називається gatsby-image змінює розмір зображень під час створення, і це означає, що менші екранні пристрої, такі як смартфони, не повинні завантажувати зображення великого розміру. Зрештою, він спокійно завантажує зображення, що означає подальше збільшення швидкості вебпошуку. Це недоступно для Next.js.

2.6. Висновок до другого розділу

Системи або фреймворки активно використовуються розробниками при створенні застосунків з різним функціоналом і рівнем складності. За наявності великого різноманіття систем з різними властивостями і потребами існує необхідність у зборі та аналізі інформації про наявні засоби розробки. Аналіз інформації про існуючі фреймворки, бібліотеки та можливості їх взаємодії дасть змогу правильно підібрати інструменти для розробки конкретних проєктів, вибрати оптимальний варіант для конкретного вебзастосунку з урахуванням поставлених задач. Вибір і використання фреймворків грає важливу роль в розробці, реалізації та супроводі як простих вебзастосунків, так і складних програмних систем.

РОЗДІЛ 3
ОПИС, ІНСТРУКЦІЯ З ВИКОРИСТАННЯ ТА РЕЗУЛЬТАТИ ТЕСТУВАННЯ РОЗРОБЛЕНОГО ВЕБЗАСТОСУНКУ

3.1. Опис розробленого вебзастосунку

В ході дослідження, в рамках кваліфікаційної роботи був розроблений вебзастосунок «SL1nk» («Еслінк» від словосполучення “shorten link” – скоротити посилання). Мета створення програми – демонстрація можливостей добре підібраних інструментів розробки ПЗ на прикладі застосунку, що надає можливість великій кількості користувачів одночасно послуговуватися розробленим сервісом скорочення посилань з подальшою їх експлуатацією для будь-яких цілей і веденням аналітики переходів за посиланнями. Такий сервіс і переваги скорочення посилань несуть користь як для окремих користувачів, так і для SMM, B2C, B2B та інших видів компаній інтернет бізнесу. «SL1nk» дозволяє користувачам створювати особисті кабінети, авторизуватись, використовувати інструмент створення скорочених посилань, переглядати та редагувати списки раніше утворених посилань з відображенням детальної інформації про кожне з них а також контролювати кількість переходів за кожним скороченим посиланням. Крім того, користувачі не мають обмежень у кількості або унікальності таких посилань. Тобто, за потреби, можна створювати декілька коротких посилань для одного і того самого оригінального посилання, що надає змогу контролювати кількість переходів за посиланнями з різних каналів публікування. Це може стати у нагоді для формування більш детального аналізу трафіку клієнтів з різних рекламних напрямків. Наприклад, одне посилання користувач може розмістити на фейсбук, друге на рекламну платформу, третє запустити в інстаграм чи твіттер і т.д. Таким чином користувач отримує можливість аналізувати – яка кількість відвідувачів приходить з цих каналів. Часто рекламні кампанії використовують декілька різних креативів для зацікавлення аудиторії. Власні короткі посилання для кожного з креативів,

створені у розробленому сервісі, в тандемі з аналітикою публікацій у соціальних мережа, дозволяють аналізувати який з креативів отримав найбільший рівень конверсії, тобто кількість зацікавлених користувачів, що перейшли за посиланням по відношенню до загальної кількості переглядів публікації. Так, на одну вебадресу користувач «SL1nk» може отримати безліч скорочених посилань. Така можливість передбачена лише в деяких подібних сервісах і зазвичай не є безкоштовною. Переваги скорочених посилань:

- Зручні для сприйняття аудиторії. Часом посилання є настільки довгими, що їх публікація на тих чи інших платформах чи друкування на папері в такому вигляді виявляється неможливою. Ця користь скорочення посилань мабуть найбільш очевидна і не вимагає детального пояснення.
- Можливість відстеження трафіку з певного напрямку/публікації. Це відбувається, тому що кожна нова генерація скорочення дає новий URL, який, тим не менш, незмінно веде до заданої користувачем сторінки.

Сервіс повністю пристосований для відображення на популярних пристроях. Наступні рисунки ілюструють відображення сторінки форми скорочення посилань на мобільних (рис. 3.1) і комп'ютерних пристроях (рис. 3.2).

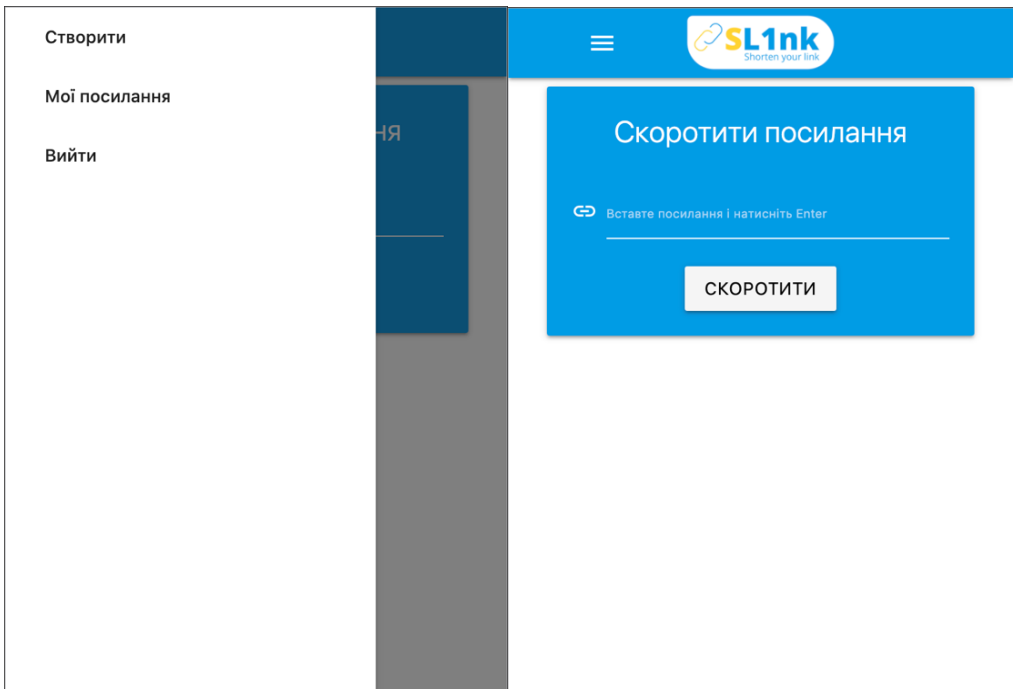

Рис. 3.1. Відображення на смартфоні

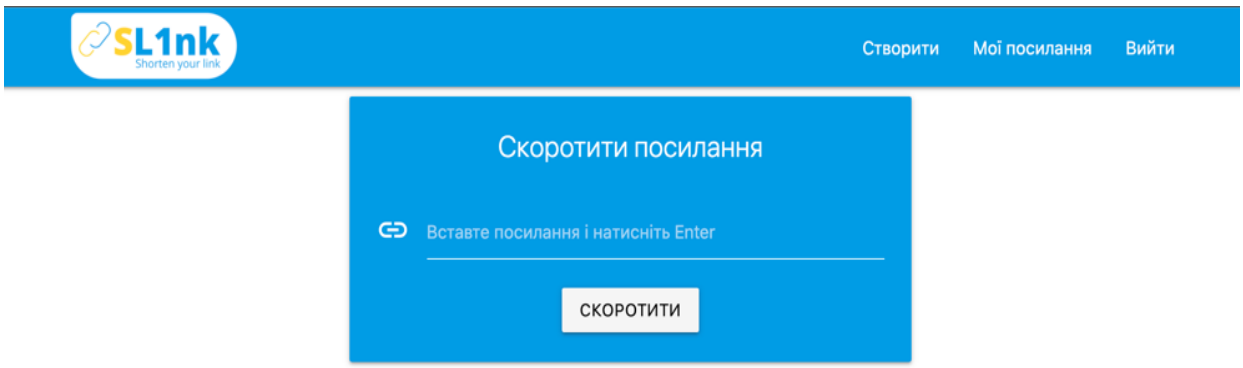

Рис. 3.2. Відображення на комп’ютері

3.2. Опис використаних технологій

Для розробки сервісу «SL1nk» були використані такі програмні каркаси, як: Node.js, React.js, Express.js а також деякі бібліотеки і пакети з npm. В якості бази даних для зберігання даних авторизації користувачів, списків їх створених посилань і кількості переходів виступає MongoDB. Таким чином розробку проведено за допомогою відомого набора інструментів розробки веборієнтованого програмного забезпечення – MERN (MongoDB, Express, React, Node). Цей набір технологій дозволяє розробникам веб застосунків писати код серверної і клієнтської частин ПЗ з використанням однієї МП – Javascript. Клієнтська частина застосунку написана на основі: компонентів, хуків стану, ефекту, контексту, маршрутизатора та інших можливостей програмного каркасу React.js, що дають змогу розробникам створювати вдало сконструйовані програми і використовувати прийоми декомпозиції коду, які суттєво покращують архітектуру складних, масштабованих програм, їх подальшу підтримку іншими розробниками та інтеграцію зі сторонніми системами. Віртуальна об’єктна модель документа, вбудована у React.js, зберігається в пам’яті і дозволяє застосунку працювати без перезавантаження всієї сторінки для оновлення елементів інтерфейсу. А маршрутизація і fetch запити дають змогу користувачам переключатись між сторінками застосунку без перезавантаження, контролюючи своє місцезнаходження в адресному полі браузера. Основна стилізація інтерфейсу програми виконана за допомогою бібліотеки Materialize.css, але в проєкті також присутня індивідуальна стилізація з метою

реалізації зручного відображення таблиць посилань та інших частин інтерфейсу на мобільних пристроях. MongoDB – це сервіс для створення і керування базами даних з відкритим вихідним кодом, що має зрозумілий інтерфейс і простий спосіб використання. Для взаємодії застосунку з БД використовувалась ODM бібліотека Mongoose, що надає рішення на основі об'єктів для моделювання схем БД. В ході розробки «SL1nk» за допомогою цієї бібліотеки були створені схеми User та Link, що моделюють збережену у MongoDB інформацію про користувача і скорочене посилання відповідно. Серверну частину вебзастосунку було написано на програмній платформі Node.js та каркасі Express.js, за допомогою можливостей маршрутизації і залучення проміжного ПЗ. Функції створення користувачів, створення посилань і додавання даних у БД, перевірки даних БД і маршрутизації запитів від скорочених посилань на оригінальні – вся ця логіка реалізована у маршрутизаторах. Для асинхронних запитів на сервер з клієнтської частини та отримання відповідей задіяний API Fetch. В «SL1nk» дані користувачів надійно захищені методами шифрування, використанням унікальних ключів, що звуть токенами, генеруються при авторизації і потрібні для ідентифікації користувачів. У реалізації системи захисту особистих даних використана бібліотека JSON Web Token. Коли користувач натискає «Вхід», сервер отримує запит з емейл адресою і паролем користувача, звіряє їх з інформацією у БД і якщо такий користувач існує в системі – сервер створює унікальний токен, в який закодовано інформацію про користувача, і відправляє його на сторону клієнта. Цей токен зберігається в локальному сховищі браузера, має обмежений час життя, що у «SL1nk» закінчується через годину після авторизації і слугує пропуском користувача в систему. Програма розкодовує токен і звіряє його з даними кожного разу як здійснюється запит на сервер. Таким чином на стороні клієнта не зберігаються особисті дані як то пошта чи пароль. Для генерації скорочених посилань використовується бібліотека ShortID. З метою побудови гнучкого дизайну, для полегшення подальших модифікацій і створення підґрунтя для масштабування вебзастосунку використано шаблон проєктування і розробки ПЗ MVC. Модель життєвого циклу програми –

каскадна. За методологією цього шаблону та прийомів декомпозиції код програми розділено на логічні структури – моделі, контролери і блоки представлення.

3.1 Інструкція з використання

Користувачу, що вперше заходить на вебсторінку застосунку спершу буде запропоновано зареєструватись та увійти. Тут варто сказати що присутня перевірка даних з форми сервером, отже є деякі правила. Пароль має бути утворений не менш як з 6 символів, а електронна пошта – мати стандартний формат. В разі введення неправильних даних чи вдалої реєстрації користувача буде сповіщено (рис. 3.3).

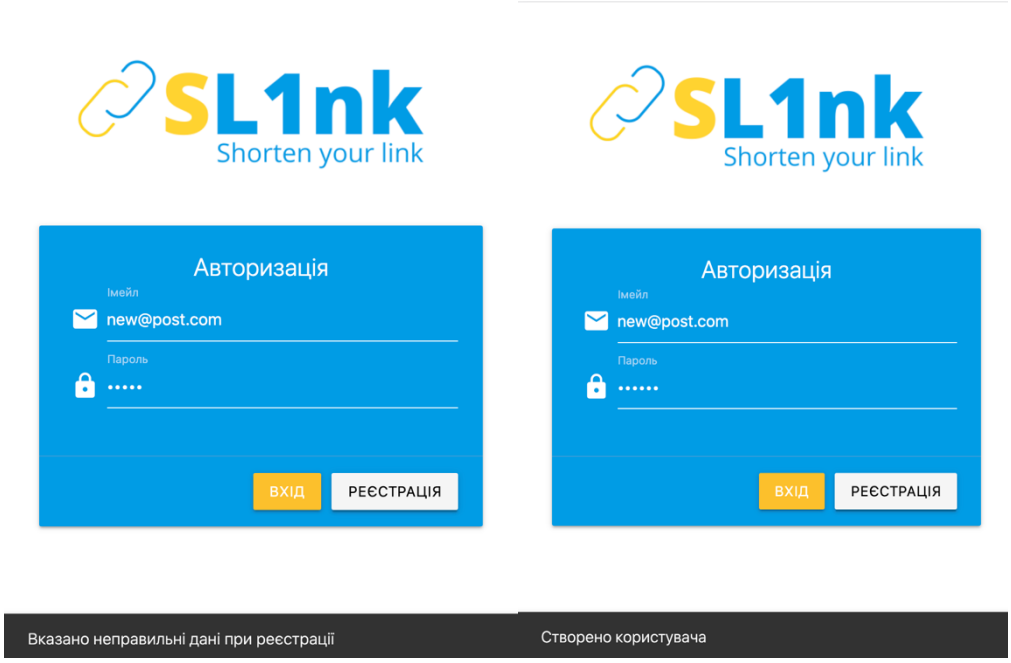

Рис. 3.3. Реєстрація

Користувачу, що вже зареєструвався необхідно увійти з вказаними раніше поштою та паролем. В разі введення не правильних даних чи успішної авторизації користувача буде сповіщено (рис. 3.4). Авторизований користувач автоматично потрапляє на сторінку створення скороченого посилання (рис. 3.4).

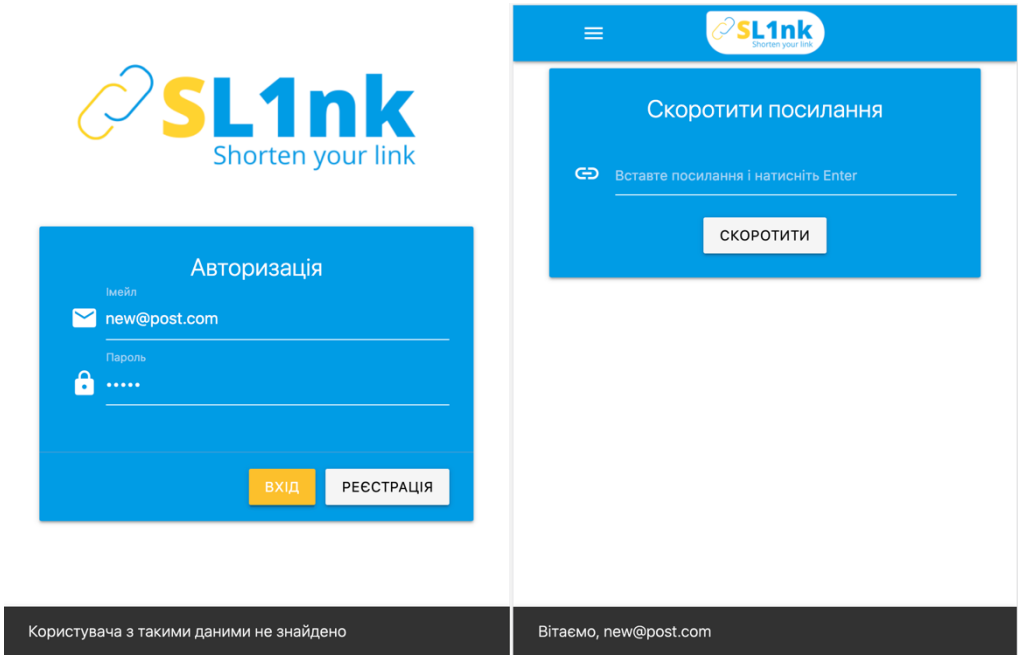

Рис. 3.4. Авторизація

Користувач, що збирається створити посилання має вставити скопійоване посилання у розумне поле і натиснути кнопку Скоротити чи клавішу Введення. Поле названо розумним тому, що воно перевіряє чи вказав користувач код протоколу передачі даних «https://». Якщо ні – поле автоматично підставить необхідний код оригінальному посиланню (рис. 3.5). Якщо посилання вказано вірно, ніяких дій не потребується і система просто створить скорочене посилання, після чого користувача буде автоматично направлено на сторінку детальної інформації про скорочене посилання (рис. 3.5). Саме на сторінці «Деталі посилання» присутній калькулятор переходів за скороченим посиланням. Кожного разу, як будь-хто в мережі інтернет переходить за скороченим посиланням, в полі «Кількість переходів за посиланням» збільшується число переходів. Для відображення оновленої інформації необхідно лише перезавантажити сторінку чи перейти на іншу сторінку застосунку і повернутися назад. Також на цій сторінці додано дату створення скороченого посилання (рис. 3.6).

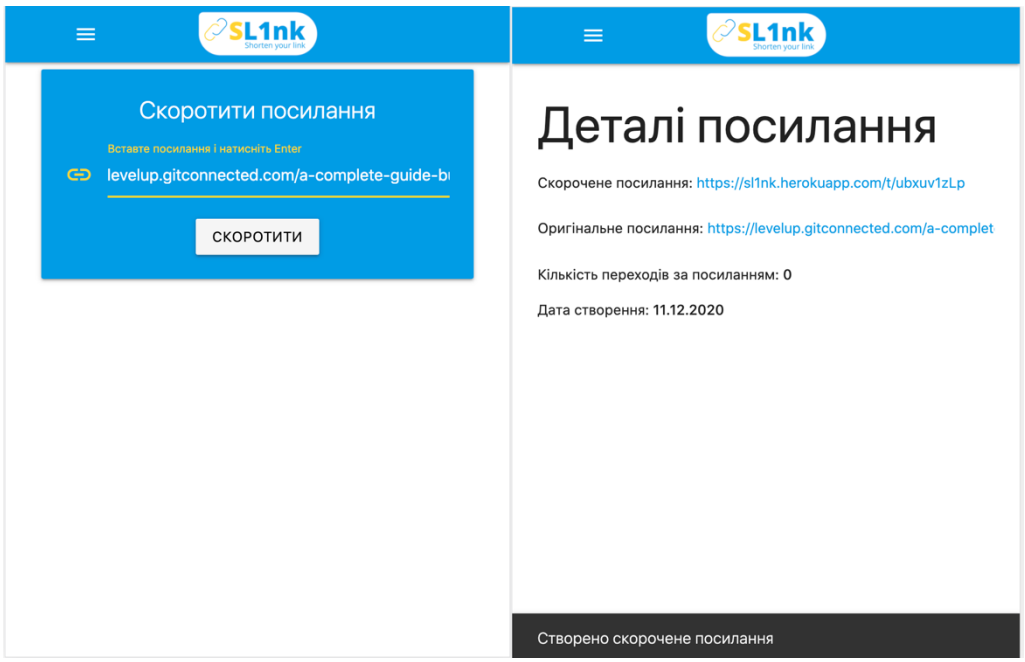

Рис. 3.5. Скорочення посилань

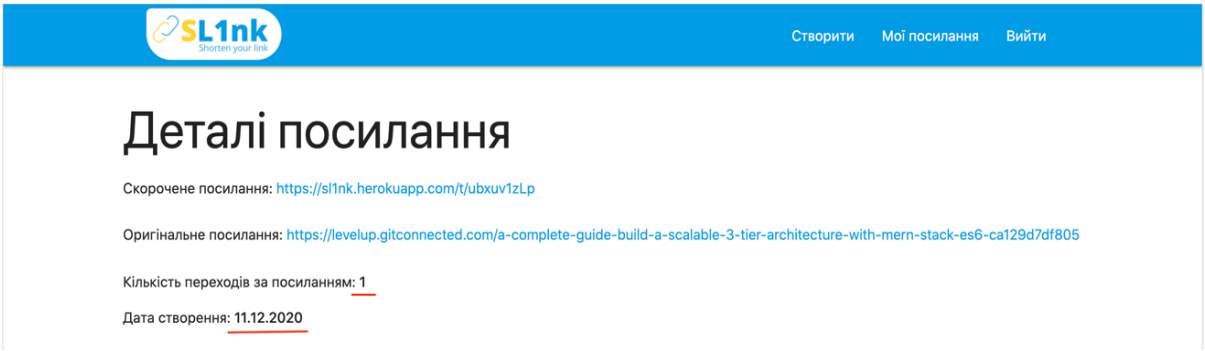

Рис. 3.6. Деталі посилання

Для перегляду раніше створених посилань в «SL1nk» існує пункт меню «Мої посилання». Тут відображені всі раніше створені користувачем посилання у вигляді таблиць. Кожне посилання має кнопки «Деталі» та «Смітник» (рис. 3.7, 3.8). Якщо натиснути кнопку «Деталі» – відкриється сторінка «Деталі посилання». Якщо натиснути кнопку «Смітник» – виникне вікно підтвердження перед видаленням посилання. Після схвалення у вікні підтвердження сторінка списку оновиться. В таблиці раніше створених посилань за умови відображення на комп’ютерних пристроях присутня можливість сортування за параметрами: номер, коротке посилання і оригінальне посилання. Щоб відсортувати таблицю за одним з параметрів потрібно натиснути на заголовок колонки (рис. 3.7).

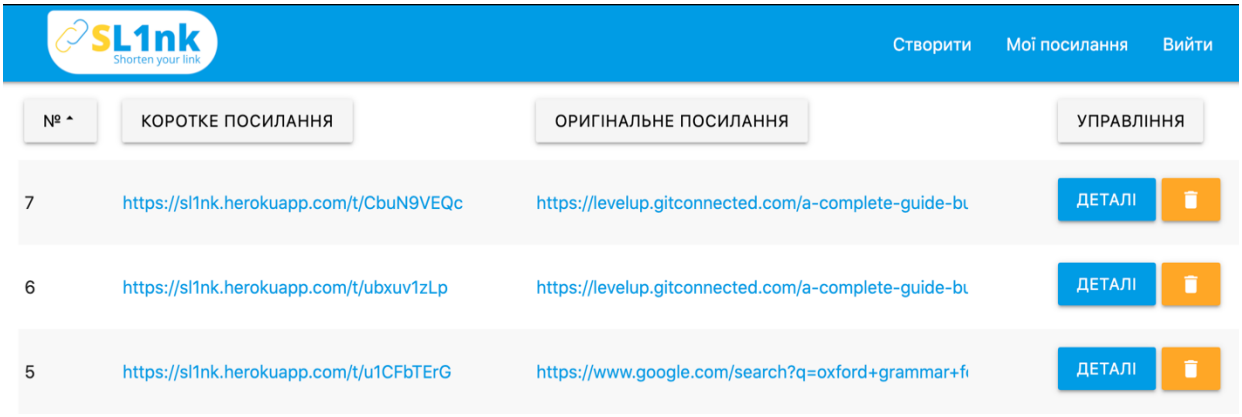

Рис. 3.7. Мої посилання

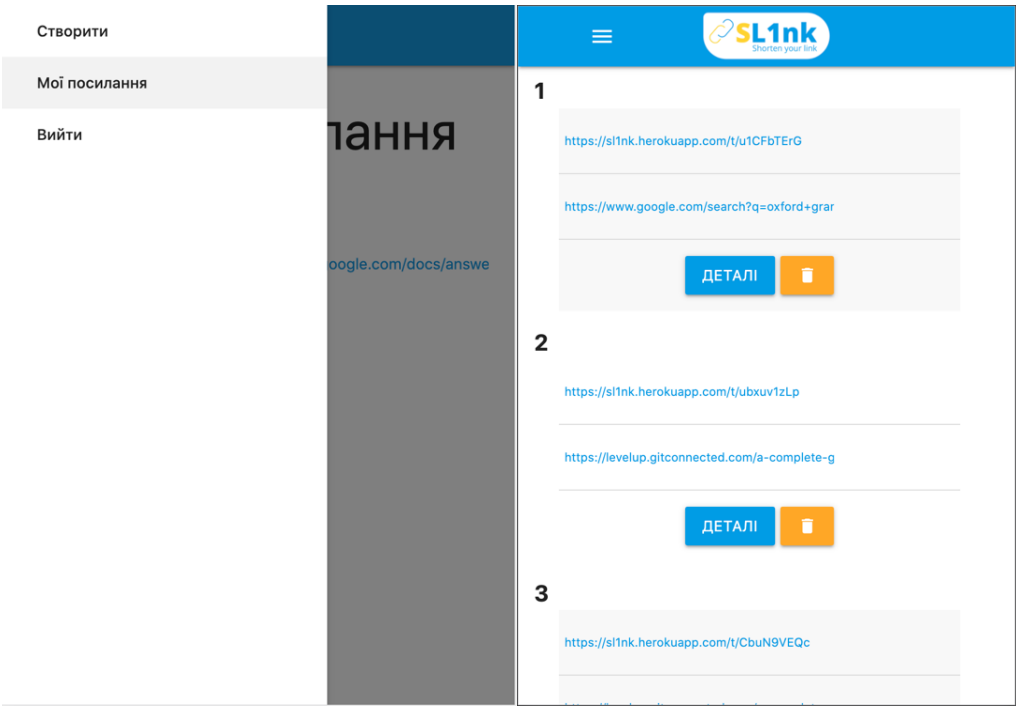

Рис. 3.8. Мої посилання (мобільні пристрої)

Для того щоб вийти з особистого кабінету – потрібно натиснути кнопку «Вийти». Коли термін дії токена закінчується – програма вийде з кабінету користувача при наступному запиті на сервер і сповістить його про необхідність нової авторизації. Інтерфейс застосунку «SL1nk» простий і інтуїтивно зрозумілий, тому, насправді, не потребує глибокого аналізу. Але ця інструкція на даному етапі включає всі його функції.

3.3. Результати тестування

Тестування інтерфейсу. Створений у рамках кваліфікаційної роботи вебзастосунок тестувався в останніх версіях таких розповсюджених браузерів як Google Chrome v80+, Safari v14+ та Firefox v80+ на комп'ютерних та мобільних пристроях з різними пропорціями екранів. На всіх пристроях були доступні всі функції програми та елементи інтерфейсу користувачів. Що дає змогу стверджувати, що результати тестування задовільні. Відзначити варто лише незручність використання html таблиць у розробці подібних інтерфейсів за допомогою HTML та CSS через таку їх властивість, як неможливість обмеження мінімальної ширини та висоти менше ніж на ширину і висоту внутрішніх елементів. Таким чином доводиться обмежувати внутрішні елементи, додаючи блоки обгортки і тому подібне.

Тестування серверу. З метою вимірювання продуктивності серверної частини вебзастосунку було проведено декілька тестів за допомогою програми Apache Bench з різними вхідними параметрами.

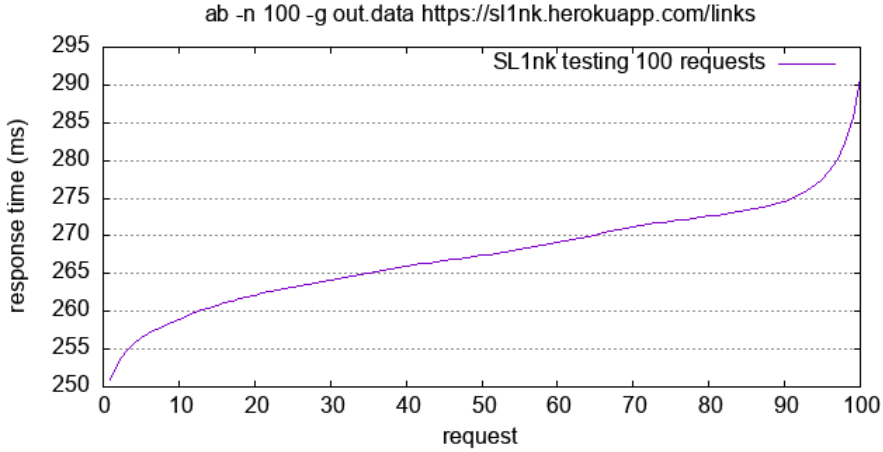

Рис. 3.9. Тест сервера 1

Перший тест на 100 почергових запитів показав, що сервер в середньому обробляв 3 – 3,8 запити/секунду (рис. 3.9). Другий тест на 100 запитів з 10 одночасних запитів пройшов успішно і показав обробку 27,45 запитів/секунду (рис. 3.10) і загальний час обробки 100 запитів зменшився з 26,3 секунд у

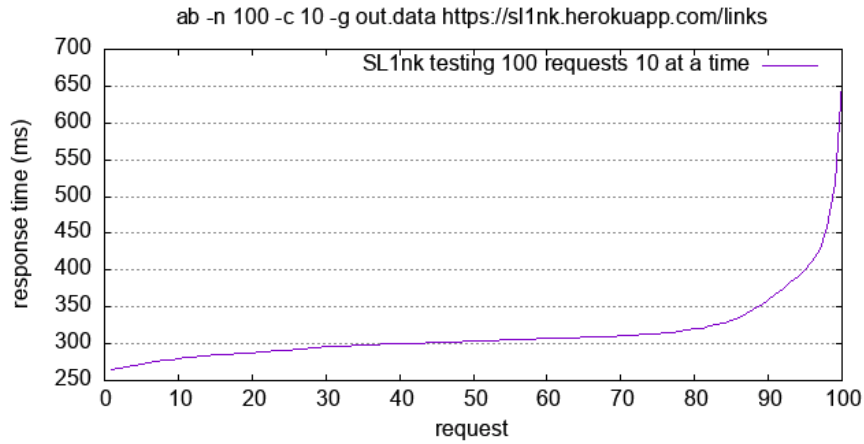

Рис. 3.10. Тест сервера 2

першому тесті до 3,64 секунди у другому. Це вказує на те, що серверу вдається в рази швидше обробляти об'єднані запити. У третьому тесті кількість запитів – 1000, а одночасних – 100. На графіку третього тестування видно що сервер зміг обробити 80% запитів доки швидкість обробки не почала стрімке падіння через завантаженість. Цього разу сервер відповідав із швидкістю 52,58 запити/секунду а загальний час обробки всіх запитів склав 19,02 секунди (рис. 3.11). І тут варто наголосити, що вебзастосунок був розміщений в мережі інтернет за допомогою безкоштовних планів MongoDB та Heroku. Було проведено і четвертий тест на продуктивність серверної частини на 1000 запитів при 200 об'єднаних. Він показав невеличке прискорення до 53,95 зап./сек., але загальний час обробки 1000 запитів зменшився на 0,49 сек. і склав 18,53 сек.

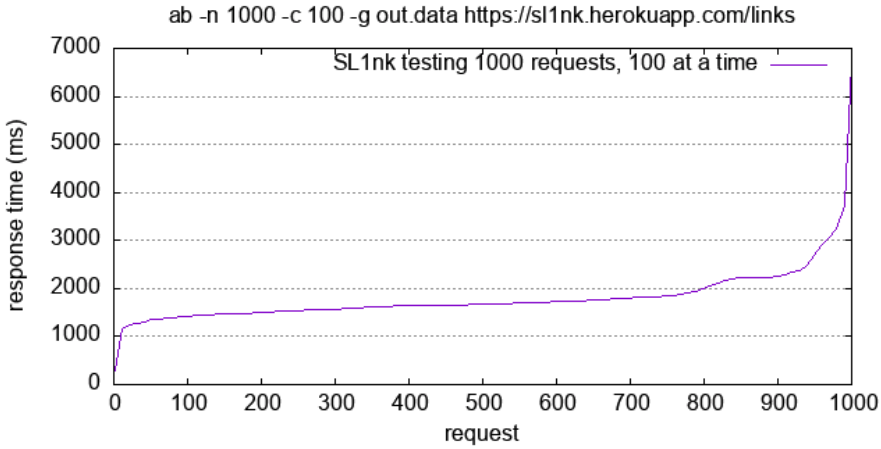

Рис. 3.11. Тест сервера 3

Наведені дані демонструють, що не дивлячись на обмеженість безкоштовних планів хостингу і БД, серверна частина «SL1nk», написана за допомогою комплексу інструментів розробки ПЗ MERN, витримує навантаження до 1000 запитів при 100 і більше одночасних. Варто відзначити, що ці показники здебільшого залежать від конкретного спорядження на якому розміщується сервер і вільно масштабуються за необхідності. Це лише питання фінансових можливостей проєкту.

Тестування швидкості завантаження сторінок. Для тестування швидкості візуалізації сторінок використано інструмент розробника «Продуктивність» вбудований у браузер Google Chrome. Цей інструмент дозволяє глибоко аналізувати швидкість завантаження вебдокументів. Тест проводився на швидкісному інтернет з'єднанні 100 мбіт/с. Компьютерний засіб – Macbook Pro з параметрами:

- Процесор – 2,6 GHz 6-ядерний Intel Core i7.
- Оперативна пам'ять – 16 ГБ 2667 MHz DDR4.
- Відеокарта – AMD Radeon Pro 5300M 4 ГБ.

Результат першого тестування демонструє, що час повного завантаження сторінки авторизації менше 200 мілісекунд, за умови, що сервер не перевантажено (рис. 3.12). Друге тестування показує час авторизації і завантаження сторінки скорочення посилань. На це вистачило 88 мс (рис. 3.13). Варто відзначити, що тести проводилися на комп'ютерній системі з великою обчислювальною потужністю і розробленому вебзастосунку не обчислюється великих об'ємів даних. Тести проводилися з кабінетом користувача із невеликим списком створених посилань – 30. Але подальші тестування показали, що навіть 50 і 100 доданих посилань не створять користувачу жодних незручностей. Результат третього тестування показує швидкість відображення сторінки створених посилань. Сторінка завантажувалась 305мс (рис. 3.14).

На даному етапі проєкту, можливий зріст завантажуваних даних залежить хіба що від кількості створених посилань. З цього можна зробити висновок, що час, необхідний для відображення сторінок для окремого користувача залежить

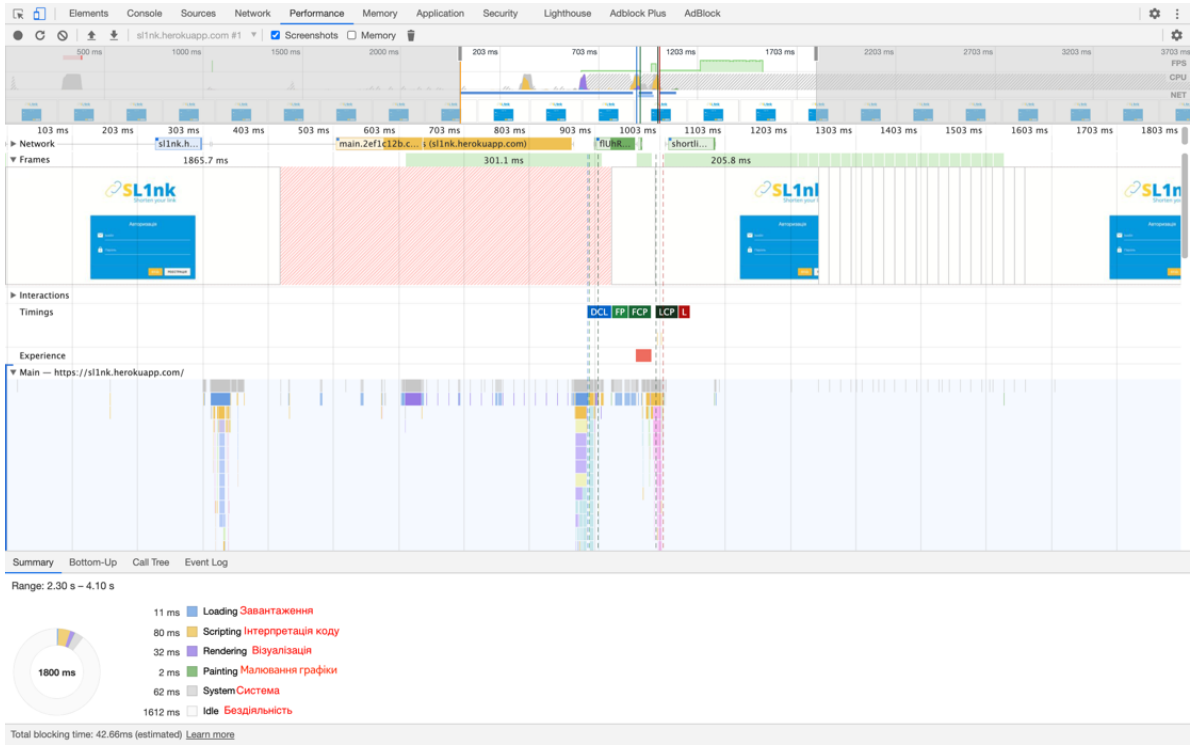

Рис. 3.12. Тест швидкості завантаження 1

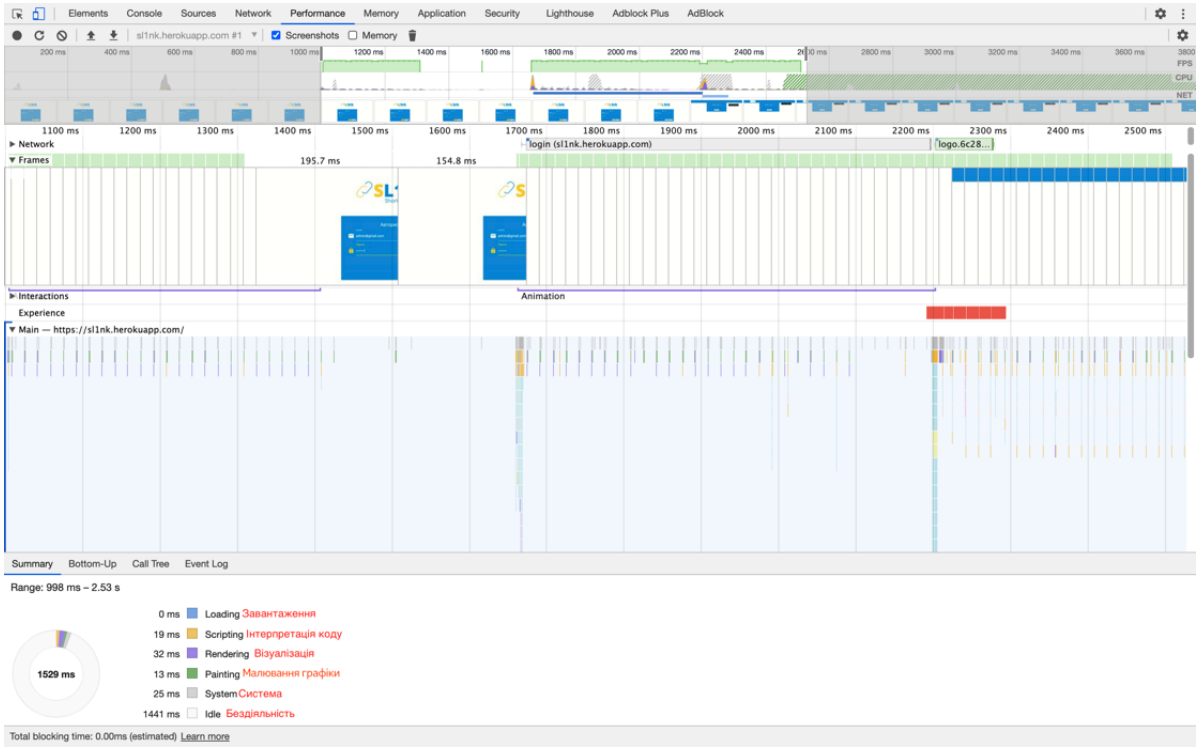

Рис. 3.13. Тест швидкості завантаження 2

більше від швидкості інтернет зв’язку користувача а також швидкості роботи сервера і БД, на яких розміщений проєкт, аніж конкретних характеристик використаної обчислювальної системи.

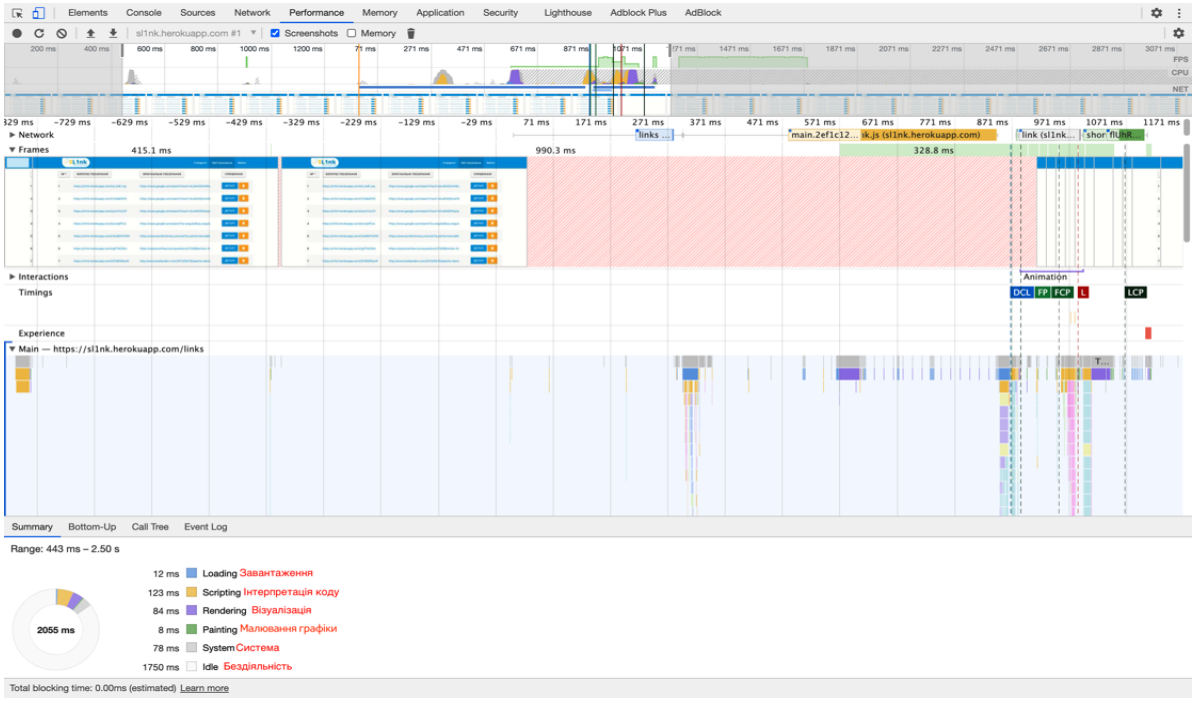

Рис. 3.14. Тест швидкості завантаження 3

3.4. Висновки до третього розділу

Розробка з використанням комплексу інструментів MERN має свої переваги:

- Дозволяє fullstack розробку.
- Основний будівельний блок React – це компонент, що підтримує стани, а також відтворюється самостійно. Розбиття програми на компоненти дозволяє розробникам зосередитися на бізнес логіці та обґрунтуванні програми. Компоненти взаємодіють зі своїми підкомпонентами шляхом обміну інформацією про стан у вигляді властивостей лише для читання, а з батьківськими – зворотніми викликами.
- Оскільки React може працювати на сервері, його код можна використовувати як у браузерах, так і на серверах. Це означає, що є можливість створювати сторінки на сервері, якщо завгодно.
- Усі технології MERN є безкоштовними і мають відкритий сирцевий код. Крім того, MERN Stack підтримується величезною спільнотою розробників, в яких можна питатися поради.
- Стек MERN підтримує архітектуру MVC, що робить процес розробки вільним.

Результати проведених тестувань демонструють швидкість роботи сервера на основі Node.js. Використання шаблону проєктування ПЗ MVC дозволило створити злагоджену архітектуру вебзастосунку SL1nk і зберігає можливість масштабування, додавання нової бізнес логіки і т.п. Ретельне тестування застосунку показало добрі результати – він працює як слід і не має очевидних можливостей зламати логіку розробленої системи.

РОЗДІЛ 4
ЕКОНОМІЧНИЙ РОЗДІЛ

При розробці програмного забезпечення важливими етапами є визначення трудомісткості розробки ПЗ, розрахунок витрат на створення програмного продукту і аналіз ринку збуту розробленого програмного забезпечення.

4.1. Визначення трудомісткості проведення дослідження та розробки необхідного для його проведення програмного забезпечення

Задані дані:

1. Передбачуване число операторів – 970.
2. Коефіцієнт складності програми – 1,5.
3. Коефіцієнт корекції програми в ході її розробки – 0,4.
4. Годинна заробітна плата програміста, грн/год – 75.
5. Коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,1.
6. Коефіцієнт кваліфікації програміста – 1,4.
7. Вартість машино-години ЕОМ, грн/год – 10.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{ ,людино} - \text{годин}, \quad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

$t_и$ – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

$t_п$ – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q * C * (1 + p), \tag{3.2}$$

де q – передбачуване число операторів;

С – коефіцієнт складності програми;

р – коефіцієнт корекції програми в ході її розробки.

$$Q = 970 * 1{,}5 * (1 + 0{,}4) = 2037, \textit{людино} - \textit{годин} \tag{3.3}$$

Витрати праці на вивчення опису задачі $t_и$ визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q * B}{(75 \ldots 85) * k}, \textit{людино} - \textit{годин}, \tag{3.4}$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

$$t_u = \frac{2037 * 1{,}1}{75 * 1{,}4} = \frac{2240{,}7}{105} = 21{,}34, людино - годин \quad (3.5)$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20 \dots 25) * k}, людино - годин, \quad (3.6)$$

$$t_a = \frac{2037}{22 * 1{,}4} = 66{,}14, людино - годин \quad (3.7)$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20 \dots 25) * k}, людино - годин \quad (3.8)$$

$$t_n = \frac{2037}{24 * 1{,}4} = 60{,}63, людино - годин \quad (3.9)$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4 \dots 5) * k}, людино - годин, \quad (3.10)$$

$$t_{отл} = \frac{2037}{4 * 1,4} = 363,75, людино - годин \quad (3.11)$$

- за умови комплексного налагодження завдання:

$$t_{отл}^{k} = 1,4 * t_{отл}, людино - годин. \quad (3.12)$$

$$t_{отл}^{k} = 1,4 * 363,75 = 509,25, людино - годин \quad (3.13)$$

Витрати праці на підготовку документації:

$$t_{д} = t_{др} + t_{до}, людино - годин, \quad (3.14)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{15..20 * k}, людино - годин. \quad (3.15)$$

$$t_{др} = \frac{2037}{15 * 1,4} = 97, людино - годин \quad (3.16)$$

$t_{до}$ – трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0,75 * t_{др}, людино - годин \quad (3.17)$$

$$t_{до} = 0,75 * 97 = 72,75, людино - годин \quad (3.18)$$

$$t_д = 97 + 72{,}75 = 169{,}75, людино - годин \quad (3.19)$$

Тепер розрахуємо трудомісткість ПЗ:

$$t = 72 + 21{,}34 + 66{,}14 + 363{,}75 + 169{,}75 = 692{,}98, людино - годин. \quad (3.20)$$

4.2. Витрати на створення програмного забезпечення для проведення дослідження

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{по} = З_{зп} + З_{мв}, грн. \quad (3.21)$$

Заробітна плата виконавців визначається за формулою:

$$З_{зп} = t * С_{пр}, грн, \quad (3.22)$$

де: t – загальна трудомісткість, людино-годин;

$С_{пр}$ – середня годинна заробітна плата програміста, грн/година

$$З_{зп} = 692{,}98 * 75 = 51973{,}5 грн. \quad (3.23)$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{мв} = t_{отл} * C_{мч}, грн, \quad (3.24)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$C_{мч}$ – вартість машино-години ЕОМ, грн/год.

$$З_{мв} = 363{,}75 * 10 = 3637{,}5, грн. \quad (3.25)$$

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУП.

$$К_{по} = 51973{,}5 + 3637{,}5 = 55611\ грн. \quad (3.26)$$

Очікуваний період створення ПЗ:

$$Т = \frac{t}{B_k * F_p}, міс, \quad (3.27)$$

де B_k – число виконавців;

$F_р$ – місячний фонд робочого часу (при 40 годинному робочому тижні $F_р$=176 годин).

$B_k = 1$

$$Т = \frac{692,98}{1 * 176} = 3,94, \quad міс \tag{3.28}$$

Таким чином, трудомісткість розробки програмного забезпечення становить 3, 94 міс.

4.3. Маркетингові дослідження ринку використання результатів дослідження

Ефективність роботи серверу і використані у розробці клієнтської частини засоби можуть впливати на швидкість завантаження вебзастосунків, сайтів та їх матеріалів. За рівнем впливу на просування продукту чи сервісу, швидкість завантаження не поставити на один рівень з бізнес логікою, інтерфейсом чи зручністю користування, але вона є важливою складовою досвіду користувачів. І тому, швидкість є одним з базових показників будь-якого успішного сервісу.

Існує багато термінів, що визначають час, необхідний для завантаження вебресурсу, наприклад: швидкість завантаження, швидкість сайту, швидкість завантаження сторінки, час відгуку, швидкість відображення даних і навіть затримка завантаження. Дебора Розен і Елізабет Пурінтон в опублікованому 2004го року дослідженні на тему вебдизайну пов’язують швидкість завантаження з простотою ресурсу. Так само Джон Моркес і Джейкоб Нільсен у роботі 1997 року стверджували, що користувачі хочуть отримати свою інформацію швидко, вони прагнуть короткого часу відгуку на гіпертекстові посилання, і в той же час вони люблять добре організовані сайти, що полегшують пошук важливої інформації. На думку цих авторів, простота дизайну робить сайт більш привабливим. Сучасні дослідження, що проводять великі корпорації та окремі дослідники вказують на те, що час завантаження і досі має значущий вплив на бажання користувачів залишатися довше на вебресурсі, що в свою чергу, має прямий вплив на рівень конверсії. Згідно досліджень Google,

проведеними в 2018 році, 53% мобільних користувачів залишають сайт, що завантажується довше 3х секунд. Те саме дослідження показало, що середній час завантаження сайтів в мережі становить 15 секунд. Показники різних досліджень дещо відрізняються, наприклад у дослідженні того ж року компанії MachMetrics, що спеціалізується на моніторингу швидкості вебсайтів, швидкість більшості ресурсів становить 8-11 секунд для повного завантаження. Але навіть це в 3 рази більше очікувань користувачів. А для сайтів електронної комерції, при завантаженні на мобільний телефон, середня позначка необхідного часу взагалі лежить близько 22 секунд. У наш час більшість великих корпорацій постійно покращує свої сервіси з метою зменшення затримок для користувача і збільшення показників конверсії. Останні дані у вільному доступі наголошують, що:

- За кожні 400 мс зменшення часу відповіді вебсторінок Yahoo побачили збільшення трафіку на 9%.
- Збільшення швидкості завантаження сторінки на 2% дають Google на 2% більше трафіку.
- Збільшення прибутку на 1% за кожні 100 мілісекунд швидкості вебсторінок показав аналіз Walmart та Amazon.
- Для мобільних пристроїв Amazon отримав збільшення конверсій на 27% за одну секунду швидкості сервісу.

Ці результати, сервісів одних з найпопулярніших компаній, показують, що збільшення прибутків шляхом зменшення часу затримки вебсторінок цілком реальне. А для менш розвинених компаній, показники швидкості мають потенціал принести навіть кращі результати у відсотковому відношенні.

Діючим засобом підвищення швидкості вебресурсу є вибір правильного набору інструментів для побудови. Використання оптимальних фреймворків для передачі та отримання даних дає змогу оптимізувати швидкість передачі та структуру застосунку чи сайту.

Перевагами використання результатів досліджень, здобутих у ході виконання кваліфікаційної роботи, є правильний вибір набору інструментів для

створення не ускладненої архітектури вебзастосунків і сайтів, що дасть змогу підвищити швидкість завантаження систем в середньому на 36% та зменшити об'єм даних для завантаження кожної окремої сторінки в середньому на 52%. Такі показники зменшення часу відповіді вебресурсу можуть суттєво вплинути на рівень задоволення користувачів, та дієздатність системи при великому навантаженні. Що, згідно результатів досліджень, позитивно вплине на загальний рівень зацікавленості продуктом чи сервісом. Знання про існуючі фреймворки, бібліотеки та можливості їх взаємодії дасть змогу правильно підібрати інструменти для розробки конкретних проєктів, що позитивно вплине на побудову злагодженої архітектури, яка матиме позитивні наслідки у подальшій розробці і неодмінно зменшуватиме вартість розробки таких систем не менше ніж на 50% а також істотно знизить ризик необхідності перебудови системи при масштабуванні.

4.4. Оцінка економічної ефективності впровадження програмного забезпечення

У результаті виконання кваліфікаційної роботи було створено вебзастосунок SL1nk, що дозволяє окремим користувачам і компаніям скорочувати посилання а також вести аналітику кількості переходів за посиланням і створювати декілька скорочених посилань для одного оригінального посилання. Наведені функції сервісу є популярними серед підприємств різного масштабу на сьогоднішній день. Окрім того, що скорочення посилань покращує якість візуального відображення, це також спрощує комунікацію споживачів з брендом. А функції аналітики і створення декількох посилань стануть корисними для компаній та користувачів, що хочуть відстежити рівні зацікавленості аудиторії рекламними публікаціями. Цей проєкт розроблений з метою демонстрації можливостей правильного вибору інструментів розробки ПЗ і не планувався як бізнес проєкт. Але, за необхідності,

таку систему можна розвинути, додавши інструменти монетизації сервісу, такі як реклама, платні плани користувачів для оренди різних функцій сервісу. Також існує можливість додавання більш розвинених інструментів аналітики трафіку, керування посиланнями, як-то зміна скороченого посилання чи заміна домену та іншої бізнес логіки задля збільшення зацікавленості користувачів сервісом. На даному етапі, розроблений застосунок не має шляхів монетизації.

ВИСНОВКИ

В ході кваліфікаційної роботи була досліджена інформація з відкритих джерел про наявні сучасні рішення, фреймворки, програмну платформу Node.js, Git, їх взаємодію і методи застосування для побудови успішної архітектури вебзастосунків та програм. В результаті, ми вказали на об’єктивні причини виникнення платформи Node.js, як фреймворка Javascript, принципи її побудови, сильні та слабкі сторони. Виокремили основні фреймворки і описали їх функції та властивості, сильні та слабкі сторони і місця застосування, порівняли їх за різними параметрами і зробили попередні висновки щодо кожного з них. Такий аналіз дасть змогу новачкам отримати базові знання про наявні рішення розробки і області їх застосування для подальшого вибору правильних інструментів для розробки проєктів зі злагодженою архітектурою та механізмами роботи. У даній роботі не акцентувалася увага на детальному аналізі кожного окремого інструменту і його можливостей, тому що їх існує велика кількість і кожен має свої особливості, про які може бути відомо лише тим, хто з ними працює досконало. В цій роботі зібрано базову інформацію про найбільш використовувані технології веброзробки. За допомогою комплекса інструментів full-stack розробки MERN розроблено вебзастосунок для динамічної обробки і скорочення URL посилань – SL1nk. Створений застосунок відповідає новітнім тенденціям розробки і безпеки даних користувачів і слугує демонстраційним прикладом правильно підібраних інструментів і злагодженої архітектури.

На наш погляд, зібрана у роботі інформація і проведене дослідження, будуть корисні для початківців, студентів, аспірантів та всіх, хто цікавиться веброзробкою. Інформація, зібрана у роботі, допоможе сформувати початкове розуміння області і теоретичні знання технологій. А розроблений застосунок – слугуватиме практичним прикладом.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Gizas A., Christodoulou S., Papatheodorou T. Comparative Evaluation of Javascript Frameworks // Proceedings of the 21st Annual Conference on World Wide Web Companion. – 2012. – P. 513-514.
2. Graziotin D., Abrahamsson P. Making Sense out of a Jungle of Javascript Frameworks, Towards a Practitioner-friendly Comparative Analysis // Lecture Notes in Computer Science. – 2014. – P. 334- 337.
3. Бодров М.Ю. Вебдодатки як еволюційний розвиток Web // Зб. наукових праць за матеріалами III Міжнародної науково-практичної конференції «Теоретичні та прикладні аспекти сучасної науки». – 2014. – № 1. – С. 18-21.
4. Чертіхін А.А., Суботіна Т.А. Застосування стандартних методів і технологій управління проєктами в середовищі веброзробки // Вісник МДУП, 2016. – №1. –[Електронний ресурс] режим доступу: http://cyberleninka.ru/article/n/primenenie-standartnyh-metodov-i-tehnologiy-upravleniya-proektami-v-srede-veb-razrabotki.
5. Barkmann H., Lincke R., Löwe W. Quantitative Evaluation of Software Quality Metrics in OpenSource Projects // Proceedings of International Conference on Advanced Information Networking and Applications. – 2009. – P. 1067-1072.
6. https://medium.com/ladies-storm-hackathons/follow-this-curriculum-to-learn-full-stack-Javascript-in-six-weeks-c0f100426902
7. Херрон Д. Node.js. Розробка серверних вебдодатків в Javascript (Packt publishing Birmingham-Mumbai David Herron "Node Web Development A practical introduction to Node, the exciting new server-side JavaScript web development stack") Пер. з англ. Слинкина А.А. – М .: ДМК Пресс, 2012. – 144с.
8. Алекс Янг, Бредлі Мек, Майк Кантелон (2018) Node.js в дії. /Алекс Янг, Бредлі Мек, Майк Кантелон//Видавництво Питер.2-е видання ISBN 978-5-496-03212-4.

9. Гурін М.І., Крилова Т.А. Інтернет-додаток на платформі Node для електронної біржі перевезень // Праці БГТУ. Серія 3: Фізико-математичні науки та інформатика, 2016. – №6 (188). [Електронний ресурс] режим доступу: http://cyberleninka.ru/article/n/internet-prilozhenie-na-platforme-node-dlya-elektronnoy-birzhi-perevozok.
10. Вільна енциклопедія Вікіпедія. – [Електронний ресурс]. URL: https://ru.wikipedia.org/wiki/Node.js.
11. Майк Кантелон, Марк Хартер, Натан Райлих, TJ Головайчук «Node.js в действии»/Издательский дом Питер/2014 г.//ISBN 13 : 978-5-496-03212-4
12. ХавербекеМ. Выразительный JavaScript.Современное веб-программирование. /3-е издание. -Питер:Издательский дом «Питер»/ 2019. //ISBN 978-5-4461-1226-5
13. Kirupa Chinnathambi / Чиннатамби Кирупа. Learning React / Изучаем React./Переводчик: Райтман М.А. Издательство: Эксмо/ 2019 // ISBN: 978-5-04-098028-4
14. Ethan Brown/Итан Браун./Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript./Питер:Издательский дом «Питер»/ 2016. –336 с.//ISBN 978-5-496-02156-2
15. Дэвид Хэррон. Node.js Разработка серверных веб-приложений на JavaScript./ДМК-Пресс/2014// ISBN: 978-5-94074-976-9, 978-1-849515-14-6
16. Roy Derks, Gaetano Checinski./ Fullstack GraphQL. The Complete Guide to Building GraphQL Clients and Servers./ Newline/2020./
17. Каскиаро М., Маммино Л./Node.js. Design Patterns/Шаблоны проектирования Node.JS./ Москва:ДМК-Пресс/2017 –396 //ISBN: 978-5-97060-485-4
18. Офіційний опис та документація фреймворку React [Електронний ресурс] режим доступу: https://reactjs.org/
19. Офіційний опис та документація Node.js. [Електронний ресурс] режим доступу: https://nodejs.org/uk/docs/
20. Офіційний опис та документація Svelte.js. [Електронний ресурс] режим доступу: https://ru.svelte.dev/docs/

21. Офіційний опис та документація GraphQL. [Електронний ресурс] режим доступу: https://graphql.org/
22. Офіційний опис та документація Express.js. [Електронний ресурс] режим доступу: https://expressjs.com/ru/
23. Офіційний опис та документація microsoft: Приступая к работе с веб-платформами Node.js в Windows. [Електронний ресурс] режим доступу: //docs.microsoft.com/ru-ru/windows/nodejs/web-frameworks
24. Актуальность JavaScript литературы. [Електронний ресурс] режим доступу: https://www.cyberforum.ru/javascript-beginners/thread2450503.html
25. Справочники веб-разработчика с примерами кода. Made with Material for MkDocs.[Електронний ресурс] режим доступу: https://xsltdev.ru/
26. Rosen, D.E. and Purinton, E.: Website design: viewing the web as a cognitive landscape. In: Journal of Business Research 57, 787–794 (2004)
27. Morkes, J. and Nielsen, J.: Concise, Scannable, and Objective: How to Write for the Web (1997) Available at: http://www.useit.com/papers/ webwriting/writing.html

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

Залежності з package.json (dependencies)

```
"dependencies": {
    "bcrypt": "^5.0.0",
    "config": "^3.3.3",
    "express": "^4.17.1",
    "express-validator": "^6.7.0",
    "jsonwebtoken": "^8.5.1",
    "materialize-css": "^1.0.0-rc.2",
    "mongoose": "^5.11.3",
    "cross-env": "^7.0.3",
    "shortid": "^2.2.16"
  },
  "devDependencies": {
    "concurrently": "^5.3.0",
    "nodemon": "^2.0.6"
  }
```

Модуль app.js сервер на Express

```
const express = require('express')
const config = require('config')
const path = require('path')
const mongoose = require('mongoose')

const app = express()
app.use(express.json({ extended: true }))

app.use('/api/auth', require('./routes/auth.routes'))
app.use('/api/link', require('./routes/link.routes'))
app.use('/t/', require('./routes/redirect.routes'))

if (process.env.NODE_ENV === 'production') {
    app.use('/', express.static(path.join(__dirname, 'client', 'build')))

    app.get('*', (req,res) => {
        res.sendFile(path.resolve(__dirname, 'client', 'build', 'index.html'))
    })
}

const PORT = process.env.PORT || 5000

async function start(){
    try {
        await mongoose.connect(config.get('mongoUri'),{
            useNewUrlParser: true,
            useUnifiedTopology: true,
            useCreateIndex: true
        })
app.listen(PORT, () => console.log(`App has been started on port ${PORT}...`))
    } catch (e) {
```

```
        console.log("Server error", e.message)
        process.exit(1)
    }
}
start()
```

Модель Link.js

```
const {Schema, model, Types} = require('mongoose')

const schema = new Schema({
    from: {type: String, required: true},
    to: {type: String, required: true, unique: true},
    code: {type: String, required: true, unique: true},
    date: {type: Date, default: Date.now},
    clicks: {type: Number, default: 0},
    owner: {type: Types.ObjectId, ref: 'User'}
})

module.exports = model('Link', schema)
```

Модель User.js

```
const {Schema, model, Types} = require('mongoose')
const schema = new Schema({
    email: {type: String, required: true, unique: true},
    password: {type: String, reuqired: true},
    links: [{ type: Types.ObjectId, ref: 'Link' }]
})
module.exports = model('User', schema)
```

Проміжне програмне забезпечення auth.middleware.js

```
const jwt = require('jsonwebtoken')
const config = require('config')

module.exports = (req, res, next) => {
    if (req.method === 'OPTIONS'){
        return next()
    }
    try {
        const token = req.headers.authorization.split(' ')[1]    // "Bearer TOKEN"
        if(!token){
            return     res.status(401).json({     message:     'Користувач     не
авторизований'})
        }
        const decoded = jwt.verify(token, config.get('jwtSecret'))
        req.user = decoded
        next()
    } catch (e) {
        res.status(401).json({ message: 'Користувач не авторизований'})
    }
}
```

Маршрутизатор аутентифікації auth.routes.js

```
const {Router} = require('express')
```

```
const config = require('config')
const bcrypt = require('bcrypt')
const jwt = require('jsonwebtoken')
const {check, validationResult} = require('express-validator')
const User = require('../models/User')
const router = Router()

// /api/auth/register
router.post(
    '/register',
    [
        check('email', 'Перевірте формат імейлу ****@*.com').isEmail(),
        check('password', 'Мінімальна довжина пароля 6 символів').isLength({ min:
6 })
    ],
    async (req, res) => {
    try {
        const errors = validationResult(req)
        if (!errors.isEmpty()) {
            return res.status(400).json({
                errors: errors.array(),
                message: 'Вказано неправильні дані при реєстрації'
            })
        }
        const {email, password} = req.body
        const candidate = await User.findOne({ email })

        if (candidate) {
            return res.status(400).json({message: 'Такий користувач вже існує'})
        }
        const hashedPassword = await bcrypt.hash(password, 12)
        const user = new User({ email, password: hashedPassword })
        await user.save()
        res.status(201).json({ message: "Створено користувача" })
    } catch (e) {
        res.status(500).json({ message: 'Щось пішло не так, спробуйте ще раз' })
    }
})

// /api/auth/login
router.post(
    '/login',
    [
      check('email',   'Користувача   з   такою   імейл   адресою/паролем   не
знайдено').normalizeEmail().isEmail(),
      check('password',   'Користувача   з   такою   імейл   адресою/паролем   не
знайдено').exists()
    ],
    async (req, res) => {
    try {
        const errors = validationResult(req)

        if (!errors.isEmpty()) {
            return res.status(400).json({
                errors: errors.array(),
                message: 'Вказано неправильні дані при авторизації'
            })
        }
        const {email, password} = req.body
        //Login check
        const user = await User.findOne({ email })

        if (!user){
```

```
            return res.status(400).json({ message: "Користувача з такими даними
не знайдено"})
        }
        //Password check
        const isMatch = await bcrypt.compare(password, user.password)
        if (!isMatch){
            return res.status(400).json({ message: "Користувача з такими даними
не знайдено" })
        }
        const token = jwt.sign(
            { userId: user.id },
            config.get('jwtSecret'),
            { expiresIn: '1h' }
        )
        res.json({ token, userId: user.id, message: `Вітаємо, ${email}` })
    } catch (e) {
        res.status(500).json({ message: 'Щось пішло не так, спробуйте ще раз' })
    }
})

module.exports = router
```

Маршрутизатор посилань link.routes.js

```
const {Router} = require('express')
const config = require('config')
const shortid = require('shortid')
const Link = require('../models/Link')
const auth = require('../middleware/auth.middleware')
const router = Router()

router.post('/generate', auth, async (req, res) => {
    try {
        const baseUrl = config.get('baseUrl')
        const {from} = req.body
        const code = shortid.generate()

        const re = /(?:https|http):\/\//
        const matches = from.match(re)
        const newfrom = matches ? from : "https://" + from
        //const existing = await Link.findOne({ from })
        /*if (existing){
            return res.json({ link: existing })
        }*/
        const to = baseUrl + "/t/" + code

        const link = new Link({
            code, to, from:newfrom, owner: req.user.userId
        })
        await link.save()

        res.status(201).json({ link })
    } catch (e) {
        res.status(500).json({ message: 'Щось пішло не так, спробуйте ще раз' })
    }
})
router.get('/', auth, async (req, res) => {
    try {
        const links = await Link.find({ owner: req.user.userId })
        res.json(links)
    } catch (e) {
        res.status(500).json({ message: 'Щось пішло не так, спробуйте ще раз' })
```

```
    }
})
router.get('/:id', auth, async (req, res) => {
    try {
        const link = await Link.findById(req.params.id)
        res.json(link)
    } catch (e) {
        res.status(500).json({ message: 'Щось пішло не так, спробуйте ще раз' })
    }
})
router.delete('/delete/:id', auth, async (req, res) => {
    try {
        await Link.findByIdAndDelete(req.params.id)
        res.status(200).json({message: `Посилання #${req.params.id} видалено з
БД`})
    } catch (e) {
        res.status(500).json({ message: 'Щось пішло не так, спробуйте ще раз' })
    }
})

module.exports = router
```

Маршрутизатор перенаправлення посилань redirect.routes.js

```
const {Router} = require('express')
const Link = require('../models/Link')
const router = Router()


router.get('/:code', async (req,res) => {
    try {

        const link = await Link.findOne({ code: req.params.code })

        if (link) {
            link.clicks++
            await link.save()
            return res.redirect(link.from)
        }

        res.status(404).json('Посилання не знайдено')

    } catch (e) {
        message
        res.status(500).json({ message:'Щось пішло не так, спробуйте ще раз' })
    }
})

module.exports = router
```

Модуль React застосунку App.js

```
import React from 'react'
import {BrowserRouter as Router} from 'react-router-dom'
import {useRoutes} from './routes'
import { useAuth } from './hooks/auth.hook'
import { AuthContext } from './context/AuthContext'
import { Navbar } from './components/Navbar'
import { Loader } from './components/Loader'

function App() {
```

```
  const { token, login, logout, userId, ready } = useAuth()
  const isAuthenticated = !!token
  const routes = useRoutes(isAuthenticated)

  if (!ready){
    return <Loader />
  }


  return (
    <AuthContext.Provider value={{
      token, login, logout, userId, isAuthenticated
    }}>
      <Router>
        { isAuthenticated && <Navbar></Navbar>}
        <div className="container">
          {routes}
        </div>
      </Router>
    </AuthContext.Provider>
  )
}

export default App
```

React хук аутентифікації auth.hook.js

```
import { useState, useCallback, useEffect } from 'react'

const storageName = 'userData'

export const useAuth = () => {
    const [ token, setToken ] = useState(null)
    const [ ready, setReady ] = useState(false)
    const [ userId, setUserId ] = useState(null)

    const login = useCallback((jwtToken, id) => {
        setToken(jwtToken)
        setUserId(id)

        localStorage.setItem(storageName, JSON.stringify({
            userId: id, token: jwtToken
        }))
    }, [])

    const logout = useCallback(() => {
        setToken(null)
        setUserId(null)
        localStorage.removeItem(storageName)
    }, [])

    useEffect(()=>{
        const data = JSON.parse(localStorage.getItem(storageName))

        if (data && data.token){
            login(data.token, data.userId)
        }
        setReady(true)
    }, [login])

    return { login, logout, token, userId, ready }
}
```

React хук http запитів http.hook.js

```
import { useState, useCallback, useContext } from "react"
import { useMessage } from "./message.hook"
import {useHistory} from 'react-router-dom'
import { AuthContext } from "context/AuthContext"


export const useHttp = () => {
    const [loading, setLoading] = useState(false)
    const [error, setError] = useState(null)

    const message = useMessage()
    const history = useHistory()
    const auth = useContext(AuthContext)

    const request = useCallback( async (url, method = 'GET', body = null, headers
= {}) => {
        setLoading(true)
        try {
            if(body){
                body = JSON.stringify(body)
                headers['Content-Type'] = 'application/json'
            }
            const response = await fetch(url, {method, body, headers})
            const data = await response.json()

            if (!response.ok) {

                if(response.status === 401){
                    message('Час сессії минув, будь ласка авторизуйтесь')
                    auth.logout()
                    history.push('/')
                }

                throw new Error(data.message || 'Щось пішло не так, спробуйте ще
раз')
            }

            setLoading(false)

            return data
        } catch (e) {
            setLoading(false)
            setError(e.message)
            throw e
        }
    }, [message, auth, history])

    const clearError = useCallback(() => setError(null), [])

    return { loading, request, error, clearError }
}
```

React хук повідомлень message.hook.js

```
import { useCallback } from "react"
```

```
export const useMessage = () => {
    return useCallback(text => {
        if (window.M && text){
            window.M.toast({ html: text })
        }
    }, [])
}
```

React маршрутизатор routes.js

```
import React from 'react'
import { Switch, Route, Redirect } from 'react-router-dom'
import { CreatePage } from './pages/CreatePage'
import { DetailPage } from './pages/DetailPage'
import { LinksPage } from './pages/LinksPage'
import { AuthPage } from './pages/AuthPage'

export const useRoutes = isAuthenticated => {
    if(isAuthenticated){
        return(
            <Switch>
                <Route path="/links" exact>
                    <LinksPage/>
                </Route>
                <Route path="/create" exact>
                    <CreatePage/>
                </Route>
                <Route path="/detail/:id" exact>
                    <DetailPage/>
                </Route>

                <Redirect to="/create"/>
            </Switch>
        )
    }

    return(
        <Switch>
            <Route path="/" exact>
                <AuthPage/>
            </Route>
            <Redirect to="/"/>
        </Switch>
    )
}
```

Сторінка аутентифікації AuthPage.js

```
import React, {useContext, useEffect, useState} from 'react'
import {useHttp} from '../hooks/http.hook'
import {useMessage} from '../hooks/message.hook'
import { AuthContext } from '../context/AuthContext'
import logo from '../image/logo.png'

export const AuthPage = () => {
    const auth = useContext(AuthContext)
    const message = useMessage()
    const { loading, request, error, clearError } = useHttp()
    const [form, setForm]= useState({
        email: '', password:''
    })
```

```
    useEffect(() => {
        message(error)
        clearError()
    }, [error, message, clearError])

    useEffect(() => {
        window.M.updateTextFields()
    }, [])

    const changeHandler = event => {
        setForm({ ...form, [event.target.name]: event.target.value })
    }

    const registerHandler = async () => {
        try {
            const data = await request('/api/auth/register', 'POST', {...form})
            message(data.message)
        } catch (e) {}
    }

    const loginHandler = async (event) => {
        if(event.key === 'Enter' || event.target.id === "loginUser")
        try {
            const data = await request('/api/auth/login', 'POST', {...form})
            message(data.message)
            // @ts-ignore
            auth.login(data.token, data.userId)
        } catch (e) {}
    }

    /*const blurHandler = useCallback(() => {

    }, [])*/

    return (
        <div className="row">
            <div className="col s12 m10 offset-m1 l6 offset-l3 center-align">
                <h1><img src={logo} alt="SL1nk" className="logo_img_big"/></h1>
                <div className="card light-blue darken-1">
                    <div className="card-content white-text">
                        <span           className="card-title           center-
align">Авторизація</span>
                        <div>
                            <div className="input-field">
                                <i className="material-icons prefix">email</i>
                                <input
                                    id="email"
                                    type="text"
                                    name="email"
                                    className="input_white"
                                    value={form.email}
                                    onChange={changeHandler}
                                    onKeyPress={loginHandler}
                                    //onBlur={blurHandler}
                                />
                                <label htmlFor="email">Імейл</label>
                            </div>
                            <div className="input-field">
                                <i className="material-icons prefix">lock</i>
                                <input
                                    id="password"
                                    type="password"
                                    name="password"
                                    className="input_white"
```

```
                                    value={form.password}
                                    onChange={changeHandler}
                                    onKeyPress={loginHandler}
                                    //onBlur={blurHandler}
                                />
                                <label htmlFor="password">Пароль</label>
                            </div>
                        </div>
                    </div>
                    <div className="card-action right-align">
                        <button
                            id="loginUser"
                            className="btn yellow darken-2 waves-effect waves-
light sign_in_btn"
                            onClick={loginHandler}
                            disabled={loading}
                        >
                            Вхід
                        </button>
                        <button
                            className="btn grey lighten-4 black-text waves-effect
waves-dark"
                            onClick={registerHandler}
                            disabled={loading}
                        >
                            Реєстрація
                        </button>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

Сторінка скорочення посилань CreatePage.js

```
import React, {useContext, useEffect, useState} from 'react'
import {useHttp} from '../hooks/http.hook'
import {useMessage} from '../hooks/message.hook'
import {AuthContext} from '../context/AuthContext'
import { useHistory } from 'react-router-dom'

export const CreatePage = () => {
    const message = useMessage()
    const history = useHistory()
    const auth = useContext(AuthContext)
    const {request} = useHttp()
    const [link, setLink] = useState('')

    useEffect(() => {
        window.M.updateTextFields()

    }, [])

    const pressHandler = async event => {
        if(event.key === 'Enter' || event.target.id === "createLink"){
            try {
                const data = await request('/api/link/generate', 'POST', {from:
link}, {
                    Authorization: `Bearer ${auth.token}`
                })
                message('Створено скорочене посилання')
                history.push(`/detail/${data.link._id}`)
```

```
            } catch (e) {}
        }

    }

    return (
        <div className="row">
            <div    className="col    s12    m10    offset-m1    l6    offset-l3
createpage_container">
                <div className="card light-blue darken-1">
                    <div className="card-content white-text">
                        <span         className="card-title         center-align
createpage_title">Скоротити посилання</span>
                        <div className="input-field">
                            <i className="material-icons prefix">link</i>
                            <input
                                id="link"
                                type="url"
                                className="input_white"
                                value={link}
                                onChange={e => setLink(e.target.value)}
                                onKeyPress={pressHandler}
                            />
                            <label htmlFor="link">Вставте посилання і натисніть
Enter</label>
                        </div>
                        <div className="center-align">
                            <button
                                id="createLink"
                                className="btn grey lighten-4 black-text waves-
effect waves-dark"
                                onClick={pressHandler}
                                >
                                Скоротити
                            </button>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

Сторінка деталі посилання DetailPage.js

```
import React, {useCallback, useContext, useEffect, useState} from 'react'
import {useParams} from 'react-router-dom'
import { useHttp } from '../hooks/http.hook'
import { AuthContext } from '../context/AuthContext'
import { Loader } from '../components/Loader'
import { LinkCard } from '../components/LinkCard'

export const DetailPage = () => {
    const {token} = useContext(AuthContext)
    const {request, loading} = useHttp()
    const [link, setLink] = useState(null)
    // @ts-ignore
    const linkId = useParams().id

    const getLink = useCallback( async () => {
        try {
            const fetched = await request(`/api/link/${linkId}`, 'GET', null, {
                Authorization: `Bearer ${token}`
```

```
            })
            setLink(fetched)
        } catch (e) {}
    }, [token, linkId, request])

    useEffect(() => {
        getLink()
    }, [getLink])

    if (loading){
        return <Loader />
    }

    return (
        <>
            { !loading && link && <LinkCard link={link} /> }
        </>
    )
}
```

Сторінка всіх посилань LinksPage.js

```
import React, {useCallback, useContext, useEffect, useState} from 'react'
import { AuthContext } from '../context/AuthContext'
import { useHttp } from '../hooks/http.hook'
import { Loader } from '../components/Loader'
import { LinksList } from '../components/LinksList'
import {useMessage} from '../hooks/message.hook'

export const LinksPage = () => {
    const [links, setLinks] = useState([])
    const [dbLinks, setDBLinks] = useState([])
    const [sortDirection, setSortDirection] = useState('ascending')
    const {loading, request} = useHttp()
    const {token} = useContext(AuthContext)
    const message = useMessage()

    const fetchLinks = useCallback( async () => {
        try {
            const fetched = await request('/api/link', 'GET', null, {
                Authorization: `Bearer ${token}`
            })
            setLinks(fetched)
            setDBLinks(fetched)
        } catch (e) {}
    }, [token, request])

    const deleteHandler = async linkId => {
        try {
            const deleteConfirmed = window.confirm("Видалити посилання?")
            if (deleteConfirmed) {
                await request('/api/link/delete/' + linkId, 'DELETE', {}, {
                    Authorization: `Bearer ${token}`
                })

                message('Видалено скорочене посилання')
                setLinks(links.filter(link => link._id !== linkId))
                return
            }
            message('Скасовано')
        } catch (e) {}
    }

    const sortHandler = async (field, e) => {
```

```
        try {
            let direction = sortDirection
            let sortedLinks = [...links]
            let sortedDBLinks = [...dbLinks]

            const descendingArrow = 'arrow_drop_up'
            const ascendingArrow = 'arrow_drop_down'

    // If the field (parameter) user sorts by is №
            if(field === 'index'){
                switch (sortDirection){
                    case 'ascending':
                        sortedDBLinks.sort((link1,   link2)   =>   (link1._id   <
link2._id) ? 1 : -1)
                    break
                    case 'descending':
                        sortedDBLinks.sort((link1,   link2)   =>   (link1._id   >
link2._id) ? 1 : -1)
                    break
                }
                setLinks(sortedDBLinks)
                return
            }

            switch (sortDirection){
                case 'ascending':
                    sortedLinks.sort((link1,    link2)    =>    (link1[field]    >
link2[field]) ? 1 : -1)
                break
                case 'descending':
                    sortedLinks.sort((link1,    link2)    =>    (link1[field]    <
link2[field]) ? 1 : -1)
                break
            }
            setLinks(sortedLinks)

        } catch (e) {console.log(e)}
    }

    useEffect(() => {
        fetchLinks()
    }, [fetchLinks])

    if (loading) {
        return <Loader />
    }

    return (
        <>
            { !loading && <LinksList links={ links } sortDirection={ sortDirection
} onSort={ sortHandler } onDelete = { deleteHandler }/> }
        </>
```

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу магістра

на тему: «Дослідження взаємодії фреймворків та програмної платформи Node.js з метою розробки високо динамічних веб-застосунків з використанням Javascript»

студента групи 121м-19-1 Єфременко Дмитра Костянтиновича

Керівник економічного розділу **Л. В. Касьяненко**

доцент каф. ПЕП та ПУ, к.е.н.

ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файла	**Опис**
Пояснювальні документи	
Кваліфікаційна_робота_Єфременко.docx	Пояснювальна записка до магістерської роботи. Документ Word.
Кваліфікаційна_робота_Єфременко.pdf	Пояснювальна записка до магістерської роботи в форматі PDF
Програма	
Program.zip	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_ Єфременко.pptx	Презентація до магістерської роботи