ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ, УПРАВЛІННЯ, ПРАВА ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Пояснювальна записка

до кваліфікаційної роботи на здобуття ступеня вищої освіти магістр

на тему: **«Програмні технології проєктування вебзастосувань»**

Виконав: здобувач вищої освіти
за освітньо-професійною програмою
Інформаційні управляючі системи та
технології спеціальності
126 Інформаційні системи та технології
ступеня вищої освіти магістр
групи 126ІСТ_мд_21
Канцібер Д. С.
Керівник: Копішинська О. П.
Рецензент: Муравльов В. В.

Полтава – 2023 року

ВСТУП

Актуальність теми кваліфікаційної роботи пов'язана з важливістю та неоднозначністю вибору програмних засобів для реалізації вебдодатків, важливість та функціональність яких стрімко розширюється. Постійна «гонка озброєнь» в галузі розробки вебдодатків призводить до появи величезної кількості інструментів. Розуміння та впровадження в роботу новітніх технологій стає критично важливим для досягнення конкурентних переваг. Як відомо, вебдодатки використовуються в різних сферах, включаючи освіту, медицину та громадську сферу, що підкреслює необхідність дослідження для вдосконалення функціональності та надійності вебдодатків у всіх соціальних, виробничих, наукових та бізнес–аспектах.

Враховуючи широкий спектр можливостей, які надають програмні засоби веброзробки та дизайну, важливо провести ретельне дослідження їх характеристик, особливостей використання, переваг та недоліків.

Зв'язок роботи з науковими програмами, темами. Робота виконана у відповідності до науково–дослідної ініціативної теми «Організаційно–методологічні аспекти впровадження інформаційно–комунікаційних систем і технологій в управлінні діяльністю сучасних організацій та підприємств за умов переходу до цифрової економіки» ДРН 0117U003099, яка реалізується на кафедрі інформаційних систем та технологій Полтавського державного аграрного університету (ПДАУ).

Метою кваліфікаційної роботи є узагальнення теоретичних та практичних знань, щодо підходів до вибору інструментів розробки сучасних вебдодатків. Для її досягнення були визначенні та реалізовані такі наукові *завдання*:

– дослідити тенденції розвитку сучасних вебтехнологій та їх вплив на підходи до розробки вебсайтів;

– розкрити сутність вебсайтів закладів освіти та їх структурних підрозділів;

– провести аналіз та вибір існуючого інструментарію веброзробника;

– розробити вебдодаток для навчально–дослідної лабораторії вебтехнологій і хмарних обчислень.

Об'єктом дослідження кваліфікаційної роботи є вибір та комбінування спеціалізованих вебтехнологій та інтегрованих програмних середовищ при проєктуванні вебдодатків та сервісів.

Предметом дослідження є: властивості і технічні аспекти застосування сучасних вебтехнологій та інтегрованих програмних середовищ.

Методи наукових досліджень: абстрактно–логічний, інформаційно–пошуковий, аналітичний, розрахунково–конструктивний, статистико–економічний.

Інформаційну базу кваліфікаційної роботи складають: наукові статті, тематична література, статистичні дані аналітичних компаній, які розміщені на вебсайтах у вільному доступі, власні спостереження та фахові літературні дослідження з досліджуваної проблеми.

Елементи наукової новизни полягають у здійсненні аналізу та оцінці впливу сучасних вебтехнологій на якісні параметри розробки вебдодатків.

Дістало подальшого розвитку:

– напрями розвитку сучасних вебтехнологій та інструментарію розробки;

– особливості вибору та використання сучасного інструментарію розробника, для створення вебдодатку структурного підрозділу закладу освіти.

Практична значущість одержаних результатів: результати роботи являють собою завершену працю з можливістю практичного використання. Застосування результатів кваліфікаційної роботи дає змогу розширити можливості технологій веброзробки вебдодатків, зокрема, для освітніх закладів. Практична реалізація вебдодатку для навчально–дослідної лабораторії дозволить підвищувати зацікавленість студентів шляхом розміщення на ньому інформаційних матеріалів, останніх новин та досягнень на базі такої лабораторії.

Апробація результатів дослідження відбувалася шляхом оприлюднення доповідей на наукових міжнародній та студентській конференціях.

Публікації. За результатами проведеного дослідження опубліковано тези доповідей: «Системи управління проєктами: порівняльна характеристика функціональних можливостей та зручність використання для бізнесу», представлені на IV міжнародній науково–практичній конференції «Сучасне підприємництво: проблеми теорії та практики» (м. Дніпро, 19 травня 2023 року); «Використання мови TypeScript при розробці вебсайтів» на I міжнародній науково–практичній конференції «Стратегічний менеджмент агропродовольчої сфери в умовах глобалізації економіки: безпека, інновації, лідерство» (м. Полтава, 28 вересня 2023 року); «Методи захисту сучасних вебсайтів від XSS–атак» на третій науково–технічній конференції «Студентська Конференція Інформаційна, Функційна і Кібербезпека» (м. Харків, 30 листопада – 1 грудня 2023 року).

Матеріали статті подано в редакцію фахового видання Computer systems and information technologies.

Структура та обсяг кваліфікаційної роботи. Пояснювальна записка кваліфікаційної роботи складається зі вступу, 3 розділів, висновків, списку використаних джерел (50 найменувань), 15 додатків. Кваліфікаційна робота містить 12 таблиць, 40 рисунків, викладена на 79 сторінках.

РОЗДІЛ 1
ПРОГРАМНО–ТЕОРЕТИЧНІ ЗАСАДИ ПРОЄКТУВАННЯ ТА РОЗРОБЛЕННЯ ВЕБДОДАТКІВ

1.1 Стан сучасних вебтехнологій та прикладні аспекти їх застосування

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

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

Очікується, що вебдодатки будуть доступні цілодобово з будь–якої точки світу та будуть адаптовані для використання з будь–якого пристрою з будь–яким розміром екрану. Окрім цього, важливими складовими сучасного вебдодатку є гнучка і масштабована система безпеки та багатий досвід користувача, побудований на клієнті, вважають автори [3].

Потрібно враховувати, що технології дуже швидко змінюються: те що було популярним вчора, сьогодні вже може бути не актуальним. Втрачають актуальність багатосторінкові сайти, адже швидкість та продуктивність таких сайтів є вкрай низькою. Це пов’язано з тим, що на кожну дію користувача запитується нова сторінка з серверу, та відбувається повне перезавантаження на клієнтській частині. На зміну їм прийшли односторінкові «сайти» Single Page Applications (SPA), які отримують сторінку з сервера лише один раз, а всі

інші операції виконуються завдяки асинхронним AJAX та JavaScript скриптам [4].

Подібні зміни вимагали нових ідей та підходів до розробки. Наприклад, поява концепції «компонентів» у веброзробці дала змогу створювати елементи, які можна повторно використовувати в різних частинах коду. Окрім цього, стали з'являтися різні бібліотеки для управління станом додатку (Redux, MobX). Обов'язковою стала можливість інтеграції вебзастосунків з різними платформами і сервісами (соціальні мережі, платіжні системи, API сторонніх сервісів). Для створення real–time застосунків, таких як чати, було створено технологію websockets, яка дозволяє встановлювати постійне двостороннє з'єднання між користувачами, без необхідності постійно створювати нові «коннекти», як при використані традиційних методів [5].

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

Для сучасного бізнесу вебсайт відіграє одну з ключових ролей. Не є перебільшенням твердження, що компанія без вебсайту приречена на провал. Завдяки вебсайту компанія може: допомогти клієнтам в здійснені оплати, залучати клієнтів до взаємовигідної співпраці, підвищувати свій імідж, аналізувати онлайн–конверсію, визначати цільову аудиторію, формувати статистику та аналітичні звіти, рекламувати свою діяльність, товар або послугу. Розглядаючи важливість вебсайтів у бізнесі, важливо враховувати, що не існує універсального набору інструментів. Різні категорії вебсайтів вимагають різних підходів до створення.

На сучасному етапі інтернет–розвитку існує велике розмаїття вебсайтів, спрямованих на виконання різних завдань та задоволення різних потреб користувачів. Графік популярності вебсайтів різного призначення зображено на рисунку 1.1. На ньому видно, що найпопулярнішими типами сайтів є чати,

соціальні мережі, пошукові двигуни та всеможливий e–Commerce: інтернет–магазини, аукціони, банкінг.

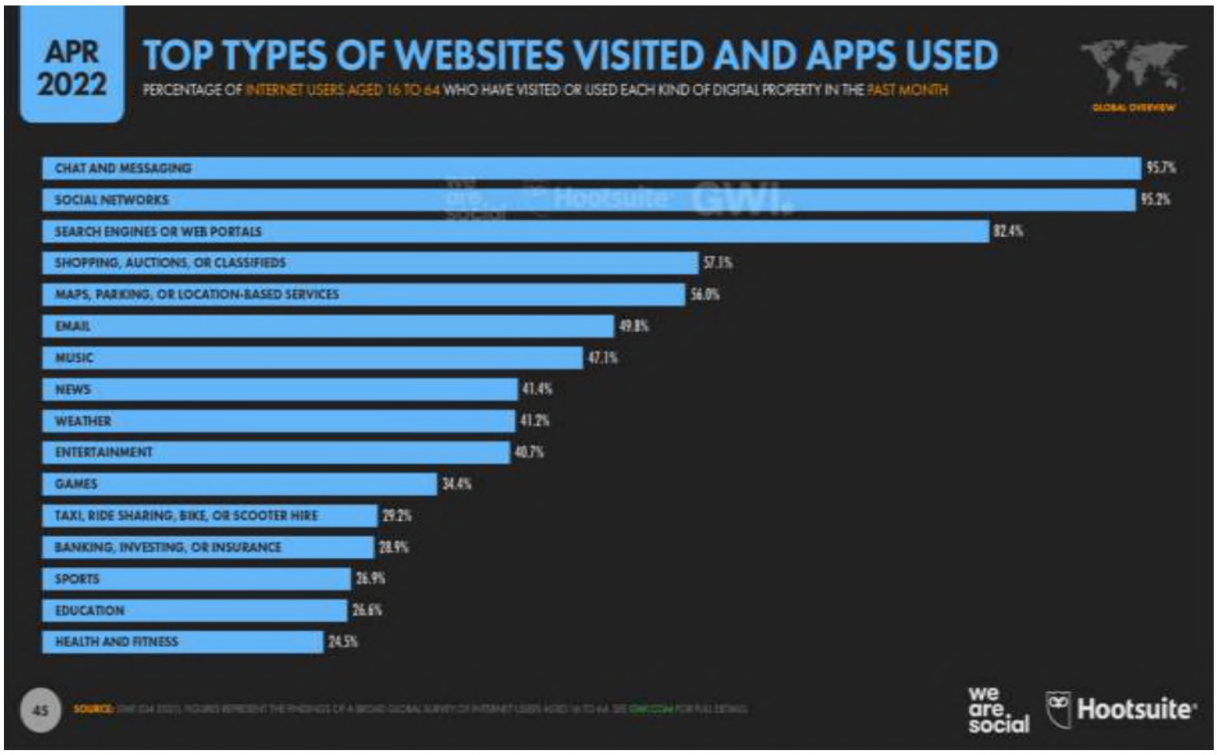


Рисунок 1.1 – Статистика відвідуваності різних типів вебсайтів [6]

Сайти можна класифікувати по–різному. Для дослідження саме в розрізі використовуваного інструментарію, їх можна умовно поділити на категорії за такими показниками як: вид діяльності, характеристика, перелік типових програмних рішень. Результати дослідження наведено в таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика категорій вебсайтів

Категорія вебсайтів	Опис	Характеристика	Типовий інструментарій
Сайти–лендінги	Односторінкові вебсайти створені з метою реклами і маркетингу. Користувачі потрапляють на такі сайти через посилання в електронному повідомленні або рекламу в соц. мережах.	Простий дизайн, чітка ідея, максимально концентрований контент.	HTML та CSS, або можна використати генератор сайтів: Wix, Jekyll, Hugo.

Продовження табл. 1.1

Інформаційні сайти	Спрямовані на надання користувачам інформації різного характеру: новинні портали, сайти компаній, освітні ресурси ті інші.	Розгалуджена структура, акцент на текстовому і мультимедійному контенті, можливість взаємодії з користувачем через зворотній зв'язок.	CMS (Content Management Systems), або ж фреймворки: Django, Ruby on Rails, Express.js.
Соціальні мережі	Платформи для взаємодії та обміну інформацією Блоги фокусуються на індивідуальному контенті, соціальні мережі – на взаємодії, форуми – на обговореннях.	Складні за будовою сайти, наявна можливість публікації та обміну контентом, профілі користувачів, механізми коментування та «лайків».	Для створення платформ використовують технології, Facebook та Instagram фреймворк React.
e–Commerce	Інтернет–магазини, майданчики оголошень (по типу OLX), сайти зі знижками інтернет–аукціони, системи онлайн–платежів.	Каталог товарів з можливістю фільтрації та пошуку, корзина покупок та система онлайн–оплати, особисті кабінети.	Для сайтів такого типу зазвичай використовуються спеціалізовані CMS: Joomla, OpenCart.
Інтерфейси масштабних проєктів	Це можуть бути корпоративні портали, системи управліннями великими даними, системи управління проєктами, банківські сайти, пошукові системи.	Складна архітектура з багатьма взаємодіючими модулями, забезпечення безпеки та управління доступом, формування аналітичних звітів.	Для розробки подібних вебсайтів використовуються потужні фреймворки: Angular, Nest, Laravel, Spring.
Інтерфейси хмарних обчислювальних систем	Вебінтерфейси для взаємодії з хмарними системами, наприклад: консолі управління хмарними платформами, такими як AWS Management Console чи Google Cloud Console.	Управління ресурсами хмарної інфраструктури, моніторинг та налаштування хмарних послуг, взаємодія з різними сервісами хмарної платформи.	Для таких систем, аналогічно з інтерфейсами масштабних проєктів найкраще використовувати потужні фреймворки.

Отже, як видно з таблиці 1.1, кожна категорія вимагає відповідного підходу та інструментарію для успішної реалізації.

Окремо важливим моментом є адаптивність. Зараз популярним є підхід «Mobile First». Автори роботи [7] закликають завжди дотримуватися цього принципу. Цей принцип означає, що застосунки повинні розроблятися таким чином, щоб в першу чергу гарно виглядати на мобільних пристроях, а вже після цього на широких екранах.

Сайт, створений за принципом Mobile First має наступні переваги: високе ранжування, швидке завантаження, сприяння здійсненню швидкої покупки. До появи таких інструментів як CSS та JavaScript такий підхід, та й взагалі адаптивність вебсторінки була неможливою. Це важливо, оскільки якщо подивитись на статистику інтернет–трафіку за 2023 рік, то можна побачити, що трохи більше ніж 60% трафіку складають користувачі мобільних пристроїв (рисунок 1.2).

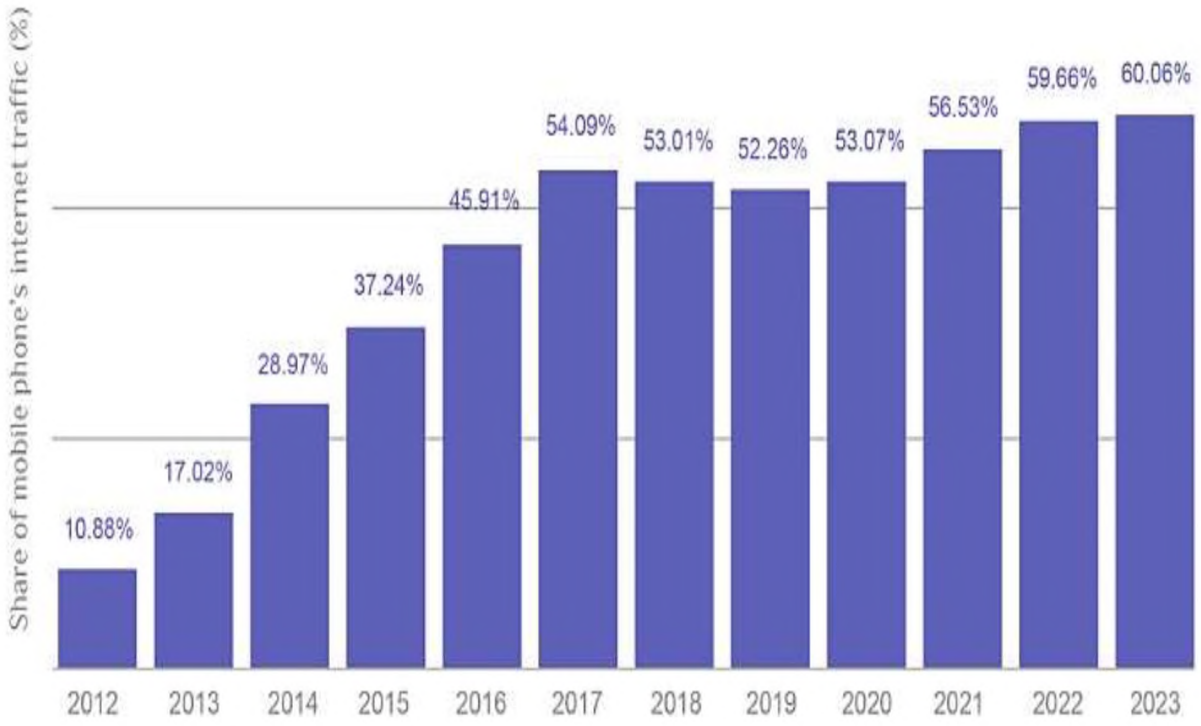


Рисунок 1.2 – Графік глобального використання інтернет–трафіку користувачами мобільних пристроїв [8]

Крім того, розвиток мобільних технологій також призводить до необхідності впровадження в розробку специфічних фреймворків та інструментів, таких як React Native, Flutter або Xamarin. За допомогою цих

інструментів розробники можуть створювати кроссплатформені додатки, які працюють як в браузерах, так і на iOS та Android. Такий підхід дозволяє писати один код, який буде виконуватись на різних видах девайсів, що суттєво економить фінансові ресурси замовника та час розробника [9].

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

1.2 Аналіз основного інструментарію для розробки вебдодатків

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

Бібліотека – комплекс модулів, функцій та компонентів та іншого «готового» коду. По суті, це збірник коду, який можна повторно використовувати в своїх проєктах [10]. Завдяки ним веброзробнику не потрібно писати з нуля такі речі як наприклад модальні вікна, каруселі зображень, слайдери і багато інших. Це значно спрощує процес розробки.

Фреймворк включає в себе бібліотеки, але також він задає «каркас» майбутнього застосунку. Зазвичай фреймворк реалізує один з паттернів проєктування, таких як: MVC (Model–View– Controller), MVP (Model–View–

Persistor), MVVW (Model–View–ViewModel). Тобто, саме фреймворк керує розробником, вказує йому як саме писати код.

Деякі фреймворки дають більше свободи, деякі менше. Вони прискорюють та спрощують процес розробки, а також забезпечують більшу безпеку, ніж самописні рішення. В сучасних фреймворках враховані всі класичні інструменти взлому [10].

Загалом, різницю між бібліотекою та фреймворком дуже добре описує поняття «Inversion of Control». Це означає, що використовуючи бібліотеку ви контролюєте де і як ви викликаєте її функції. А у випадку з фреймворком саме він контролює де та яким чином вам розміщувати свій код, а також, коли його потрібно викликати [11].

1.2.1 Інструменти Frontend–розробки

Обов'язковим для реалізації фронтенд–частини сайту є лише 1 засіб – HTML. Все інше, включаючи CSS, JavaScript та різні бібліотеки, фреймворки й препроцесори не є обов'язковим з точки зору браузеру, але в сучасному світі без подібних інструментів не створюється жоден (окрім самих простих, коли розробник тільки навчається) вебдодаток у світі.

HTML (Hypertext Markup Language) – основа будь якого сайту, його каркас. За допомогою нього описується сама структура сайту, елементи які на ньому будуть знаходитися [12]. Препроцесори дозволяють розширити функціонал HTML при роботі з верстанням. Написаний за допомогою препроцесора код після інтерпретації сервером перетворюється в звичайний HTML. Прикладом таких препроцесорів може бути Pug або Haml.

До переваг використання HTML–препроцесорів відноситься:

– простіша робота з великим об'ємом коду;

– дотримання принципу DRY (Don't Repeat Yourself);

– зменшення загального часу необхідного на розробку та підтримка організації коду [13].

Оскільки HTML дозволяє лише розміщувати елементи на сторінці, постає необхідність в інструменті, який буде вказувати браузеру як ці об'єкти

виглядають. Рішенням цієї проблеми є CSS (Cascading Style Sheets). За допомогою таблиці стилів можна задавати, розмір, колір, положення одних елементів відносно других, поведінку курсора при наведенні та багато інших.

Для CSS, так само як і для HTML, існують препроцесори, а також фреймворки. Найпопулярнішими препроцесорами для CSS є Stylus, SCSS, LESS. Загальна ідея CSS–фреймворків – допомогти побудувати гарно структурований вебсайт з можливістю подальшого розвитку, а також додати додаткові функції для покращення процесу розробки стилів. Фреймворки дозволяють створювати сучасні вебінтерфейси, але також вони накладають свою специфіку та обмеження [14].

Рейтинг популярності CSS–фреймворків наведено в Додатку А.

З графіку (див. Додаток А) видно, що найпопулярнішими рішеннями є: Bootstrap, Tailwind CSS, MaterializeCSS, Foundation, PureCSS, Ant Design, Bulma, Semantic UI/Fomant.

JavaScript – мова програмування, яка використовується веброзробниками для створення динамічних і інтерактивних елементів сайту. Існування JavaScript зробило можливим існування вебдодатків – застосунків, в яких оновлення інформації відбувається без оновлення всієї сторінки [16]. У JavaScript фактично немає конкурентів: 98% вебсайтів використовують саме цю мову. Але альтернативи все ж є: Dart, CoffeeScript, Haxe, ClojureScript. Вважати їх повноцінними аналогами не можна, адже в результаті всі вони все рівно компілюються в JavaScript код.

Окремо варто розглядати TypeScript – мову програмування, що розширює JavaScript шляхом надання можливості статичної типізації. Використання TypeScript зазвичай не є обов'язковим (за винятком деяких фреймворків, наприклад – Angular), але його використання дуже бажано, особливо в проєктах з великою кількістю класів, функцій та змінних. Розробнику, який використовує TypeScript не треба буде витрачати час на те щоб дізнатися, які поля є у об'єкта, або що повертає та чи інша функція. В цьому йому допоможе IDE, яка повністю підтримує TypeScript, та буде

показувати розробнику всю необхідну інформацію своєю функцією autocomplete (яка дуже погано працює з звичайним JavaScript) [17].

Використання стандартного JavaScript хоч і цілком можливе, але часто розробники обирають якусь готову бібліотеку, або фреймворк. Графік популярності базований на кількості питань, що стосуються відповідного фреймворку на Stack Overflow наведено на рисунку 1.3.

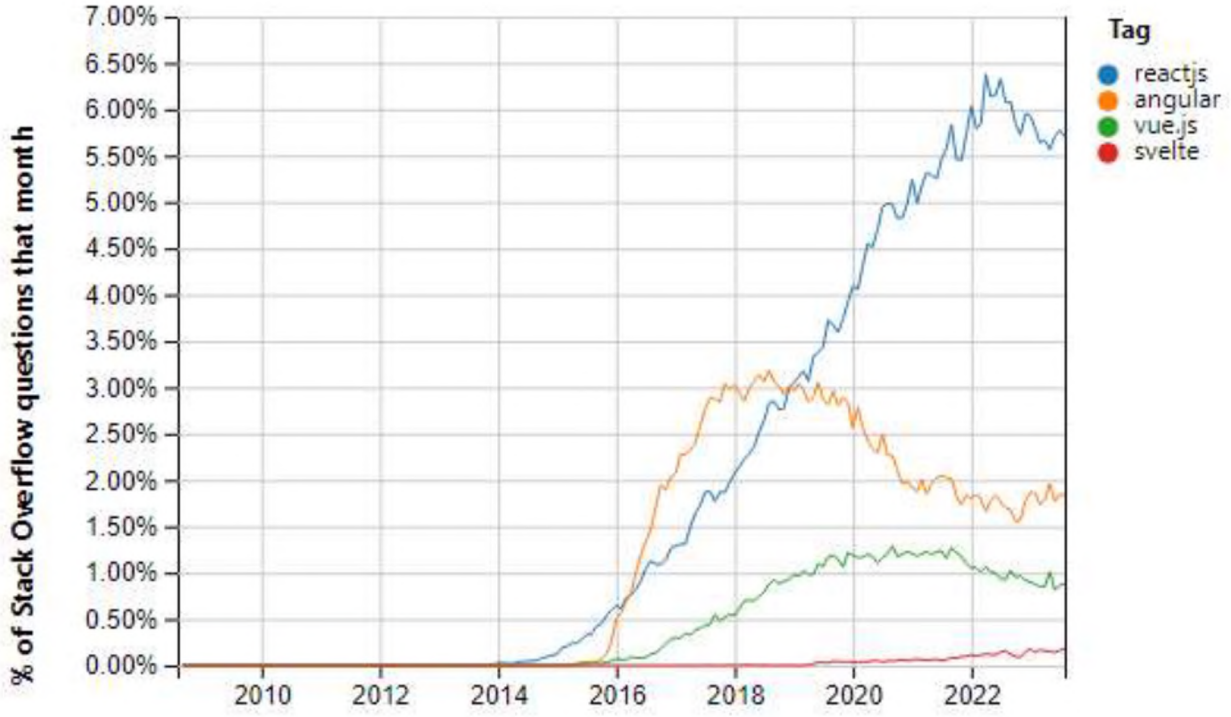


Рисунок 1.3 – Графік популярності фреймворків на Stack Overflow станом на 14.09.2023 [18]

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

1.2.2 Інструменти Backend–розробки

Клієнт спілкується з серверною частиною за допомогою HTTP–запитів. В цих запитах міститься посилання на кінцеву точку (URL–адреса на сервері, яка може приймати запити), тип запиту, та за необхідністю додаткова корисна інформація – payload. Після обробки запиту сервер повертає відповідь клієнту, яка містить в собі інформацію, що передбачена розробником. Сервер (тобто backend частина вебдодатку) повинен виконувати наступні функції: обробка HTTP–запитів, обробка бізнес–логіки, взаємодія з базою даних, авторизація та аутентифікація, логування. Вебсервер зберігає та надає доступ до контенту вебсайту – зображень, тексту, відео.

Графічне зображення класичної взаємодії клієнту (frontend) та серверу (backend) наведено на рисунку 1.4.

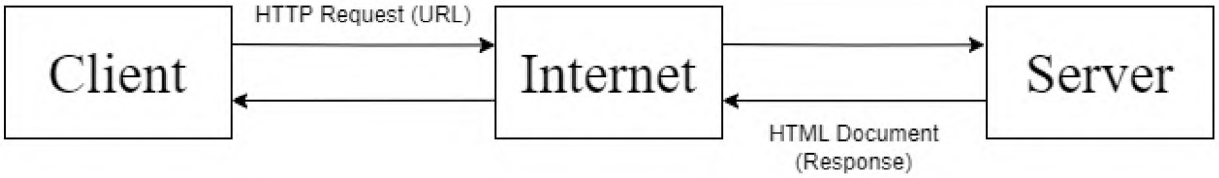


Рисунок 1.4 – Взаємодія клієнта та сервера

Хоча вебсервери зазвичай розміщують вебсайти, доступні в інтернеті, вони також можуть використовуватися для зв'язку між вебклієнтами та серверами в локальних мережах, таких як Intranet компанії. Вебсервери також можуть обмежувати швидкість відповіді окремим клієнтам, щоб не допустити перевантаження ресурсів та задовільнити запити більшої кількості користувачів [19].

Прикладами таких програмних засобів є Apache та Nginx. Результати дослідження [20] показали, що в більшості тестувань Apache виявився швидшим за Nginx, з чого можна зробити висновок, що Apache краще підходить для вебсайтів з великою кількістю запитів в хвилину. На рисунку 1.5 зображена статистика використання вебсайтами різних вебсерверів. Можна побачити, що найпопулярнішим є саме Nginx.

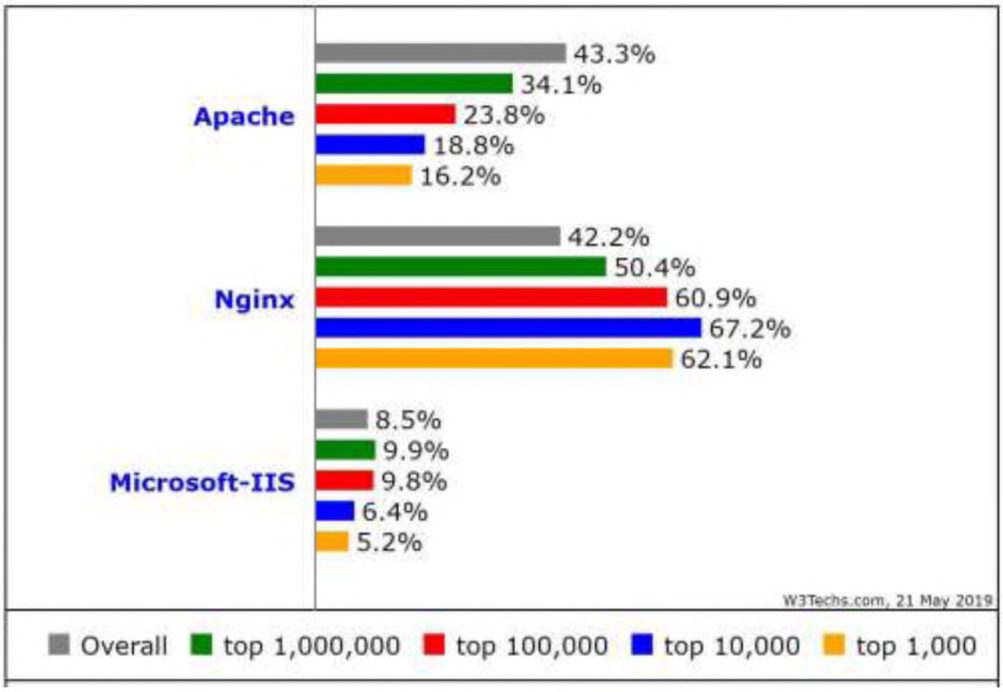


Рисунок 1.5 – Статистика використання вебсерверів [21]

Backend–частину вебсайту зазвичай пишуть на таких мовах програмування як Java, Python, Ruby, Node.js, PHP. Проте, зараз майже ніхто не використовує мову програмування саму по собі. Переважна більшість backend–розробників в своїй роботі використовують фреймворки.

Найпопулярніші backend–фреймворки станом на 2023 рік наведено на рисунку 1.6

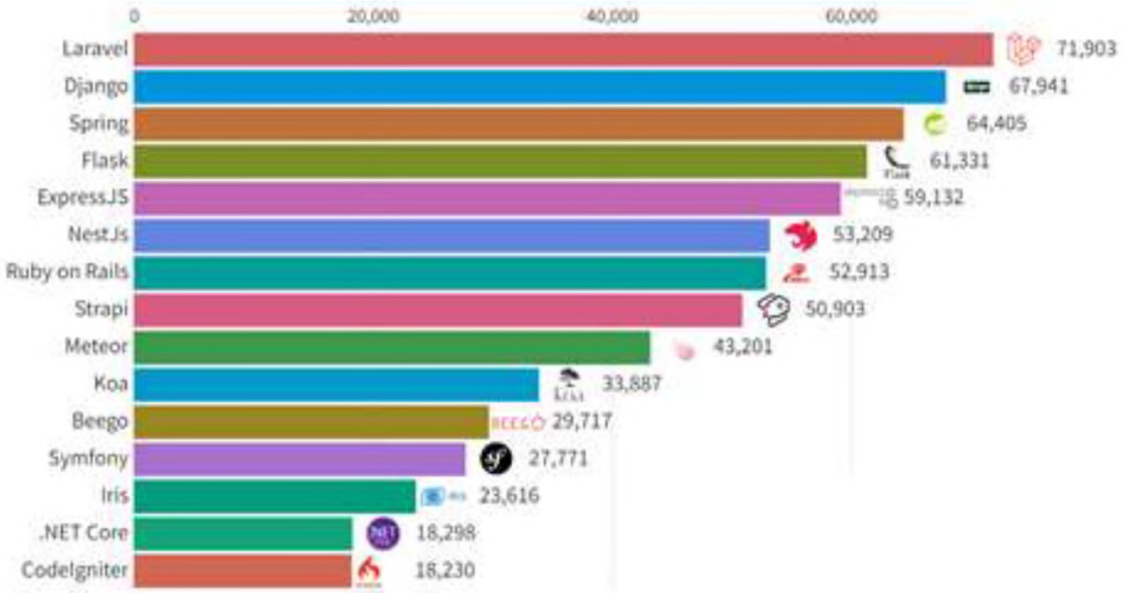


Рисунок 1.6 – Діаграма популярності backend фреймворків [22]

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

Окрім мов програмування, важливою складовою серверу є бази даних. База даних – це організована структура, призначена для зберігання, зміни й обробки взаємопов'язаної інформації. Саме в них зберігається вся інформація, пов'язана з роботою застосунку. Бази даних бувають: централізовані, хмарні, реляційні, нереляційні, об'єктно–орієнтовані, операційні та розподілені. Найпопулярнішими є реляційні, нереляційні (NoSQL) та об'єктно–орієнтовані БД.

В реляційних БД інформація зберігається у вигляді таблиць пов'язаних між собою. Реляційна БД передбачає наявність ключів. Первинний ключ використовується для встановлення унікального ідентифікатору запису, а зовнішній ключ для встановлення зв'язків між таблицями. Головна перевага реляційних баз – ACID (Atomicity, Consistency, Isolation, Durability). Цей принцип гарантує надійність та цілісність даних в процесі їх обробки.

В нереляційних БД можливо сховище виду «ключ–значення», документоорінєтоване сховище, графові бази та bigtable–подібні бази. В таких базах немає чітких зв'язків між даними та немає чіткої структури. В NoSQL використовується підхід CAP (Consistency, Availability, Partition Tolerance). Перевагою NoSQL є швидкість та продуктивність [23].

В об'єктно–орієнтованих БД інформація представлена у вигляді об'єктів, як в об'єктно–орієнтованих мовах програмування. Візуально преставляється у вигляді дерева, вузлами якого є об'єкти. Для опису властивостей (полів) об'єкту використовуються стандартні та конструйовані кориистувачем типи. Перевагою таких баз є чітка типізація та відсутність неспівпадіння моделі даних в застосунку та БД.

Для звернення до реляційних та об'єктно–орієнтованих баз даних використовуються SQL–запити – спеціальні конструкції написані за допомогою мови SQL. При використанні NoSQL баз форма запитів відрізняється в залежності від обраної БД, наприклад в MongoDB

використовується мова запитів, подібна до JavaScript – MongoDB Query Language.

Отже, реляційні БД ідеально підходять для роботи з даними, структура яких не вимагає частих змін, нереляційні – для зберігання великих об'ємів неструктурованої інформації, а об'єктно–орієнтовані БД – для проєктів, де потрібна потужна база з високою функціональністю.

Рейтинг популярності баз даних наведено на рисунку 1.7

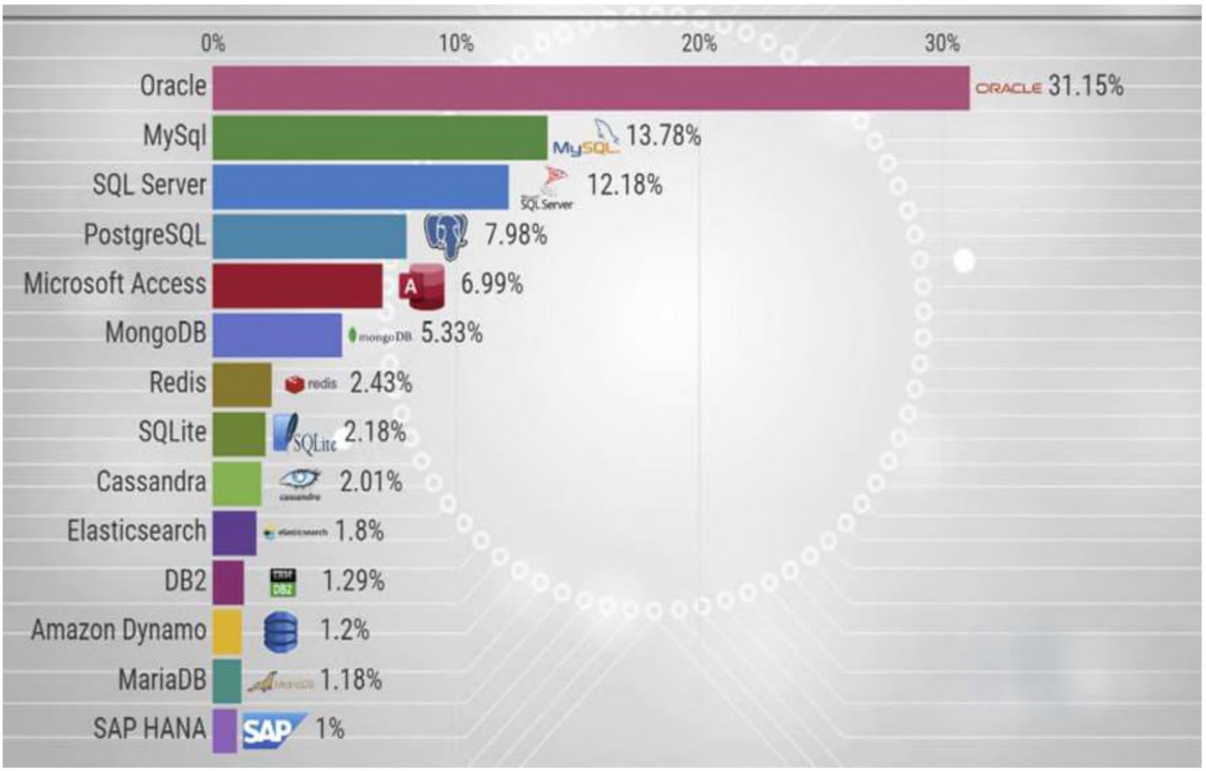


Рисунок 1.7 – Діаграма популярності баз даних [24]

Для управління базами даних використовуються системи управління базами даних (СУБД). Загальне призначення СУБД полягає в наданні централізованої системи для зручного та ефективного управління даними. Перевагами використання СУБД є: багатокористувацький доступ до БД, надійність даних, можливість створення бекапів. До недоліків відноситься: комплексність, вартість, вразливість БД, часті оновлення [25]. За видами СУБД поділяються на реляційні (RDBMS), документові (DoDBMS), стовпчасті (CDBMS). Окрім цього існують також СУБД, які спеціалізуються

на різних типах сховищ, наприклад системи управління базами даних у пам'яті (IMDBMS), а також системи управління хмарними базами даних, де постачальник SaaS (Software as a Service) відповідає за обслуговування БД, наприклад MongoDB Atlas.

Отже, є мови програмування, є бази даних, системи управління базами даних, але це не все. Серверну частину потрібно тестувати. Для вирішення таких питань існує сучасне рішення – інструменти тестування та налагодження API, або REST–клієнти. Найпопулярнішими представниками даного класу інструментів є Postman та Insomnia.

До функціоналу подібних інструментів входить:

– відправлення HTTP–запитів. REST–клієнти дозволяють легко створювати та відправляти будь–які типи HTTP–запитів. Розробник може налаштовувати параметри запитів, передавати заголовки та тіло запиту;

– тестування API. Присутня можливість створення тестів для перевірки відповідей від серверу;

– колекції. REST–клієнт дозволяє організовувати запити і тести в колекції, що спрощує управління великою кількістю запитів. Це дозволяє автоматизувати тести, а також перемикатись між різними конфігураціями середовища (тестове, розробка, продакшн) [26].

Окрім цього існують інструменти для автоматичної генерації документації API. Найвідомішим інструментом такого призначення є Swagger. Завдки формату OpenAPI він може детально визначити структуру API, його ресурси, кінцеві точки та параметри. Це створює чітку та зрозумілу основу для взаємодії з API, що полегшує розуміння його можливостей та використання.

1.2.3 Особливості CMS

Система керування контентом (Content Management Systems, CMS) – це програмне забезпечення, яке надає певний рівень автоматизації для задач ефективного управління контентом. Зазвичай CMS являє собою багатокористувацьке ПЗ на основі сервера, яке дозволяє взаємодіяти з даними в репозитарії. CMS дозволяють редакторам створювати новий контент,

редагувати існуючий і робити його доступним для інших людей. При цьому їм не треба розбиратися в програмному коді [27]. Для використання CMS не потрібно встановлювати спеціальне програмне забезпечення. Для адміністрування та редагування використовується звичайний браузер (Google Chrome або аналогічний). Інтуїтивний інтерфейс і простота роботи з системою полегшує управління сайтом і знижує подальші витрати на підтримку вебресурсу (Додаток Б).

Центральний елемент системи – ядро CMS, відіграє ключову роль у виконанні основних функцій. Його завдання включає обробку запитів, керування базою даних та забезпечення основних механізмів безпеки. Це є фундаментальною складовою, що забезпечує стабільну та роботу CMS.

Система управління контентом нероздільно пов'язана із базою даних, яка виконує роль сховища для структурованих даних. Зазвичай використовується реляційна база даних для ефективного зберігання інформації, що дозволяє забезпечити швидкий та організований доступ до великої кількості даних. Це особливо актуально, якщо враховувати, що одним з основних видів вебсайтів, на якому використовується CMS є інтернет–магазини, які вимагають особливо чіткої структуризації даних на сервері, а також потребують швидкої реакції на дії користувача.

Окрім стандартного вбудованого функціоналу, CMS зазвичай дозволяють використовувати плагіни, для вирішення специфічних задач. Наприклад готова корзина для електроного магазину, або система онлайн–оплати покупок.

За типом CMS можна розділити наступним чином:

– системи керування вебконтентом (WCM) – тип системи, що розрахований на масову доставку контенту через вебсайт. Такі CMS чудово справляються з відокремленням вмісту від зовнішнього вигляду та публікаціями на декількох каналах;

– системи керування корпоративним вмістом (ECM) – призначені для управління контентом, який не розрахований на масового користувача. Такі

системи можуть використовуватись для керування робочими документами: резюме, звіти, службові записки;

– системи керування цифровими активами (DAM) – системи спрямовані на менеджмент та керівництво цифровими активами, як–от зображення, аудіо, відео та інші. DAM добре підходить для метаданих;

– системи керування записами (RMS) – системи для управління інформацією про операції, та записами, створеними як побічний продукт бізнес–операцій (записи про продажі, контракти тощо). RMS чудово справляється з контролем доступу.

Переваги та недоліки CMS узагальнені в таблиці 1.2.

Таблиця 1.2 – Переваги та недоліки Content Management Systems.

Переваги	Недоліки
– Великий вибір різноманітних CMS; – Висока швидкість розробки; – Дешевизна; – Не потрібно вміти писати код; – Простий і зрозумілий інтерфейс; – Підтримка одночасної роботи багатьох користувачів; – Велика кількість безкоштовних готових шаблонів та рішень; – Можливість кастомізації; – Гарне тимчасове рішеня, поки повноцінний сайт в розробці; – Більшість CMS мають модульну архітектуру, а їх функціональність легко розширюється за допомогою плагинів; – Навіть безкоштовні CMS мають «технічну підтримку» у вигляді спільноти користувачів.	– Низька адаптивність (flexablilty); – Чим специфічніше завдання, тим менше шанс, що CMS підійде для цього; – Потрібен час на засвоєння конкретної CMS; – Для простих сайтів функціональність CMS, як правило, виявляється надмірною; – Може зламатися, якщо адміністратор неправильного його налаштував; – В багатьох CMS є проблеми з безпекою [28].

Зважаючи на графік популярності CMS (рисунок 1.8), WordPress є лідером з великим відривом (43.2%) і відмінно підходить для блогів та корпоративних веб–сайтів завдяки легкості використання та розширюваності. Shopify ідеально підходить для онлайн торгівлі, Joomla — для корпоративних та соціальних мереж, тоді як Drupal варто розглядати для складних та великих проєктів, орієнтованих на гнучкість та безпеку.

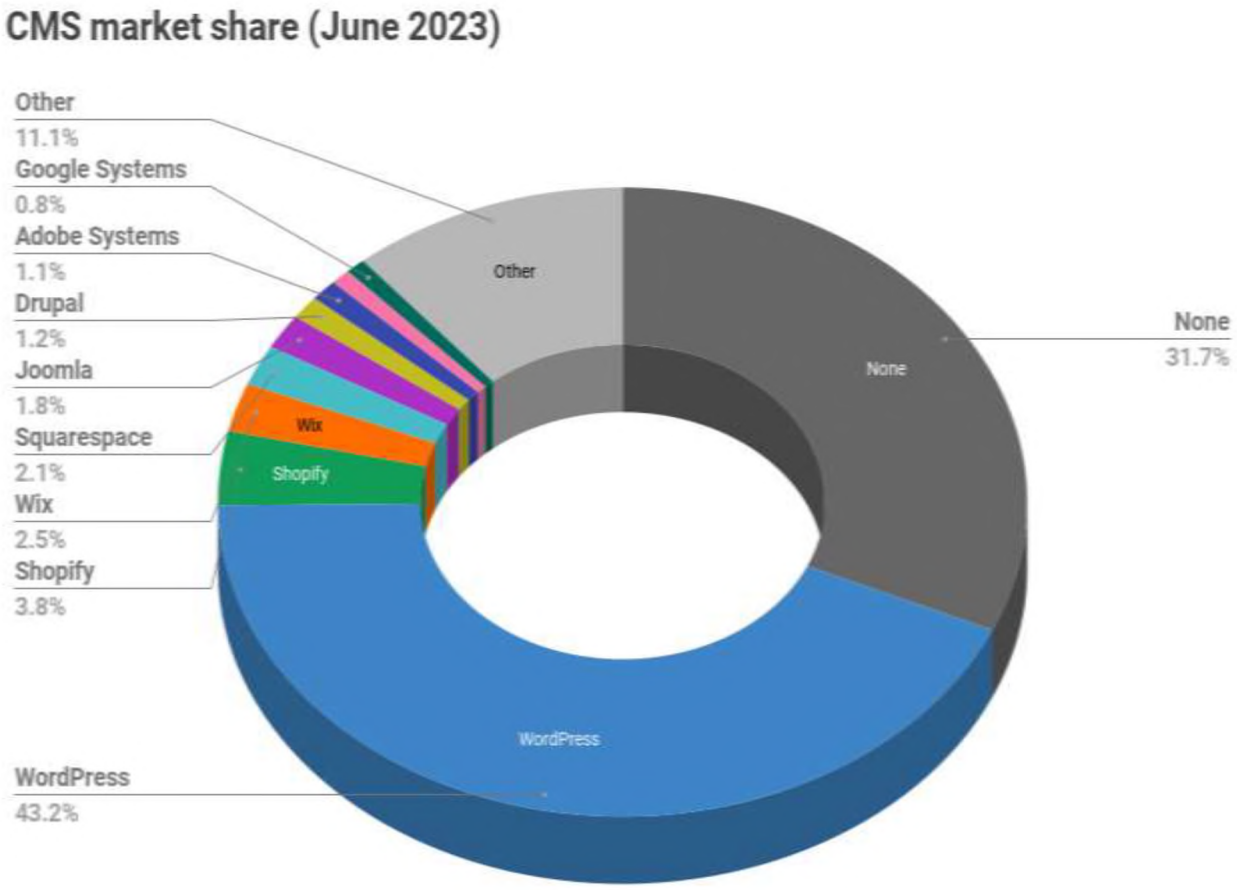


Рисунок 1.8 – Рейтинг популярності CMS [29]

Отже, можна зробити висновок, що CMS є чудовими інструментами, якщо їх використовувати в правильному руслі. Гарними прикладами вдалого використання CMS є особисті блоги, а також невеликі e–Commerce проєкти. Використовувати системи управління контентом не варто в тих випадках, коли потрібен специфічний функціонал, або є необхідність в його постійному розширені. Також варто уникати CMS, якщо йде мова про розробку масштабних проєктів, для яких важлива швидкодія та високий рівень безпеки.

1.3 Допоміжний інструментарій веброзробника

Окрім основного інструментарію, перерахованого в попередніх підрозділах, існує велике різноманіття додаткових програмних засобів, спрямованих на полегшення роботи веброзробника. До таких інструментів

можна віднести пакетні менеджери, системи контролю версій та репозитарії, редактори коду та інтегровані середовища розробки, системи автоматизації збірки, Docker та контейнеризацію. Давайте більш детально розберемся з кожним з видів інструментів.

Пакетні менеджери. В загальному, призначення пакетних менеджерів полягає в керуванні, пошуку та встановленні пакетів залежностей, а також їх оновленні. Пакетами в Node.js називають один, або декілька JavaScript–файлів, що являють собою бібліотеку. Найпопулярнішими рішеннями на даний момент є npm та yarn. За своєю структурою вони дуже схожі і надають практично ідентичний функціонал. Наразі всі менеджери пакетів отримують свої файли з реєстру пакетів NPM, тож будь–яка бібліотека, яка вам знадобиться, може бути встановлена обома рішеннями [30]. Розробники надають перевагу npm, оскільки він є вбудованим в Node.js і через це він часто є першим пакетним менеджером, з яким зіштовхується початківець.

Коли проєкт ініціалізується за допомогою одного з пакетних менеджерів, то автоматично створюється файл залежностей – «package.json». Всередині цього файлу розміщується назва проєкту, його версія, посилання на репозитарій, скрипти для запуску та відладки проєкту, а також перелік залежностей (сторонніх бібліотек). При першому запускі проєкту, створюється ще один файл «lock», в якому знаходиться перелік всіх пакетів, які потрібно встановити, а також конкретна версія кожного з них [30].

Пакети в Node.js можуть займати суттєву кількість пам’яті і при використанні репозиторіїв, постійне завантаження встановлених пакетів є неоптимальним. Пакетні менеджери вирішують цю проблему. Достатньо скачати лише основні файли проєкту та файли «package.json» і «lock», після чого запустити команду npm install або yarn install, в залежності від обраного менеджеру. В результаті буде встановлено всі пакети відповідних версій.

Наразі використання пакетних менеджерів є фактично обов’язковим при роботі з JavaScript та Node.js. Без них розробнику доведеться вручну шукати кожну бібліотеку та встановлювати її.

Системи контролю версій та репозиторії. Система контролю версій – це програмний засіб, який зберігає зміни в одному, або декількох файлах, та дозволяє в майбутньому повертатися до їх попередніх версій. Якщо ви пишете код, то використання таких систем є доцільним, оскільки вони дозволяють «відкатувати» зміни в випадку поломок, бачити хто і як змінював файли, та порівнювати зміни між різними версіями. Ці системи можна поділити на наступні категорії:

– локальні системи контролю версій. Якщо над проєктом працюєте лише ви, то найпростішим рішенням буде використання локальних систем.Однією з найбільш популярних локальних систем є RCS. Вона зберігає набори відмінностей між файлами в спеціальному форматі на диску та дозволяє в будь–який момент відтворити файл, як він виглядав в будь–який момент часу. Мінусом таких систем є схильність до появи помилок. Легко переплутати файли та зробити не ті зміни, які ви хотіли [31];

– централізовані системи контролю версій. Важливим питанням є співпраця з іншими розробниками. Для того, щоб вирішити цю проблему було створено централізовані системи. Вони мають єдиний сервер, та певну кількість користувачів, які мають до нього доступ. Прикладами таких систем є: CVS, Subversion, Perforce. До переваг можна віднести те, що кожному учаснику в тій чи іншій мірі відомо, чим займаються інші. Також адміністратор може надавати кожному учаснику доступ лише до тих файлів, з якими йому треба буде працювати. Недоліками таких систем є: відмова центрального серверу призводить до простою всієї команди розробників; при виході з ладу жорсткого диску, ви втрачаєте весь проєкт, якщо у вас не було створено резервних копій [31];

– розподілені (децентралізовані) системи контролю версій. Найсучасніший підхід до контролю версій проєкту. При такому підході, учасники отримують не лише останній знімок файлів репозитарія: натомість вони отримують повну копію та всю історію змін. Таким чином, в кожного розробника зберігається повноцінна версія проєкту і в разі виходу з ладу

серверу, з будь–якого локального репозитарію можна буде все відновити. Приклади таких систем: Git, Mercury, Bazaar, Darcs [31].

Враховуючи особливості різних видів систем, найкращим варіантом на сучасному етапі є саме розподілені системи контролю версій. На рисунку 1.9 наведено рейтинг популярності їх використання. Видно, що лідером з величезним відривом є Git.

Рисунок 1.9 – Популярність систем контролю версій [32]

Зазвичай сучасні проєкти зберігають на вебрепозитаріях, які повністю сумісні з Git. Такі сервіси дозволяють з легкістю керувати змінами в файлах проєкту. Їх основні функції полягають в запитах на зміну, сторонніх інтеграціях, клонуваннях репозитарію, ревью коду, відслідковування проблем. Прикладами таких сервісів є: GitHub, GitLab, Bitbucket.

Інтегровані середовища розробки (IDE). По–перше потрібно пояснити різницю між текстовими редакторами та IDE. Текстові редактори є легкими, а IDE – важкими, більш вимогливими до ресурсів, але в той же час надають значно більше корисних функцій. Прикладом текстового редактора саме для роботи з кодом може бути: Atom, Vim, Sublime Text, Notepad++. Такі редактори непогано підходять для невеликих проєктів, у випадку з веброзробкою – для лендінгів.

Для більш серйозних проєктів бажано використовувати спеціалізовані IDE. Вимогами до сучасних IDE є:

– підсвічування синтаксису мови програмування
– auto–complete коду
– можливість відладки коду;

– можливість встановлення плагінів;

– вбудована підтримка Git;

– підтримка препроцесорів [33].

Зазвичай, у кожної мови програмування є власні, спеціалізовані саме під неї IDE. Наприклад для Java це IntelijIDEA або Eclipse, для Python – PyCharm, для C# та C++ – Microsoft Visual Studio, для PHP – PhpStorm.

Окремо можна відзначити Visual Studio Code. По суті – це редактор, але він має таку багату екосистему плагінів, що його можна зробити повноцінним IDE для будь–якої мови програмування. Але деякі моменти будуть гірші порівняно з IDE. Наприклад, особливо слабким місцем VSC є відстуність продивнутого дебагера, що ускладнює виправлення помилок.

Системи автоматизації збірки. Зі збільшенням кількості файлів в проєктах, постала необхідність в інструментах, які будуть спрощувати збірку та компіляцію проєкта. В веброзробці існує таке поняття як bundle. Це сукупність всіх ресурсів, які є необхідними для функціонування вебдодатку. Тобто це всі html, css, js та інші файли проєкту, які перетворюються в один файл в результаті роботи автоматичних збірників. Для цього в веброзробці застосовується Webpack.

Webpack будує граф залежностей, тобто структуру проєкта у вигляді дерева, яке повинно починатись в певній точці. У випадку з Webpack таким елементом входу є ./src/index.js. Після чого він рекурсивно починає перебирати всі файли, які хоч якось залежать один від одного [34].

Часто разом з Webpack використовується Babel. Це компілятор, який перетворює сучасний JS–код в такий, який підтримується старими браузерами.

Перевагами використання систем автоматизації збірки є: пришвидшення розробки, розділення коду, вирішує проблему перезапису глобальних зміних, мініфікація (скорочення обсягу коду). Недоліками є: потрібне ретельне попереднє налаштування, можливі конфлікти між плагінами та залежностями.

Docker та контейнери. Docker – це відкрита платформа для розробки, доставки та запуску програм. Він полегшує процес розробки програмного

забезпечення шляхом упакування його в контейнери. Це дозволяє швидко встановлювати програми на будь–якій машині, уникати проблем залежностей та спрощує управління робочим оточенням.

Контейнеріація – це технологія, що дозволяє запускати застосунки з усіма залежностями, ізольовано від інших процесів [35]. Контейнеризація схожа на віртуалізацію, але різниця між ними в тому, що віртуалізація працює як окремий ПК зі своїм обладнанням, а контейнеризація не віртуалізує обладнання, отже споживає значно менше ресурсів.

Особливостями контейнерів є:

– короткий життєвий цикл. Контейнери можна з легкістю створювати, зупиняти та видаляти. Дані, що містились в контейнері теж пропадуть.

– контейнери майже не витрачають ресурси. Розмір контейнеру вимірюється в мегабайтах. Це відбувається завдяки тому, що контейнер упаковує лише необхідні для його роботи процеси та залежності;

– контейнеризація забезпечує ізоляцію процесів. Застосунки, що працюють всередині контейнера, не мають доступу до процесів ОС;

– можливість автоматизації розгортання застосунку на різних хостах, через те, що кожен контейнер зберігається в спеціальному репозиторії, в якому міститься все необхідне для запуску конкретного контейнера.

Використання Docker дозволяє вирішити наступні проблеми:

– проблему утилізації ресурсів (на одному сервері можна запускати велике число контейнерів);

– проблему залежностей та робочого оточення;

– проблему ізоляції та безпеки;

– наближення до мікросервісної архітектури.

Використання Docker вже стало стандартом при роботі над серйозними проєктами. Він значно підвищує ефективність, портативність та керованість про грамного забезпечення. Впровадження контейнеризації вирішує багато проблем, пов’язаних з розгортанням та управлінням додатками в сучасному програмному середовищі.

Висновки до розділу 1

Нами було проаналізовано сучасний стан вебтехнологій та тенденції їх розвитку. За рахунок чого ми можемо стверджувати, що наразі відбувається стрімкий розвиток вимог та інструментів для розробки вебдодатків, а також, що вебсайти є невід'ємною частиною існування сучасного бізнесу. Було проведено аналіз особливостей різних вебсайтів, та на основі цього створено таблицю, в якій вони були поділені за категоріями: за характеристиками, призначенням та оптимальним стеком технологій. Це потрібно тому що не існує універсального інструменту для кожної задачі.

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

Окрім цього було розглянуто такі сучасні інструменти як системи контролю версій, Docker та пакетні менеджери, які значно спрощують командну розробку. Це важливо, оскільки будь–який серйозний проєкт виконується групою людей, а не одним розробником.

РОЗДІЛ 2
АНАЛІЗ ТЕХНОЛОГІЙ ПРОЄКТУВАННЯ ТА КРИТЕРІЇ ВИБОРУ ДЛЯ РОЗРОБКИ ІНФОРМАЦІЙНИХ ВЕБСАЙТІВ

2.1 Аналіз вимог до функцій та структури вебсайту навчально–наукового підрозділу закладу освіти

Практичною частиною кваліфікаційної роботи є проєктування та розробка вебдодатку для одного з навчально–наукових підрозділів закладу освіти – навчально–дослідної лабораторії вебтехнологій та хмарних обчислень, створеної на базі Полтавського державного аграрного університету з метою забезпечення навчального процесу та наукових досліджень здобувачів вищої освіти ступенів бакалавра та магістра за програмою Управляючі інформаційні системи спеціальності 126 Інформаційні системи та технології, а також інших спеціальностей. Вона була створена в рамках програми розвитку навчально–матеріальної бази ПДАУ на кафедрі інформаційних систем та технологій навчально–наукового інституту економіки, управління, права та інформаційних технологій, з метою сприяння ґрунтовному вивченню навчального матеріалу.

Для реалізації поставленої цілі необхідно вирішити наступні задачі:

– вивчити особливості вебсайтів закладів освіти та структурних підрозділів;

– провести аналіз готових рішень, для виявлення основних вимог, трендів та в результаті – розроблення технічного завдання;

– визначити структуру майбутнього вебдодатку;

– провести порівняння та вибір інструментів для реалізації;

– реалізувати вебдодаток, провести тестування та розробити інструкції користувача.

В сучасному світі вебсайт є одним з ключових інструментів комунікації навчального закладу з абітурієнтами, студентами, викладачами та іншими

зацікавленими людьми. Через зростання конкуренції в галузі, утримувати позиції можна лише при використанні сучасних інструментів маркетингу, в даному випадку – вебсайтів.

Кононец Н.В. дає наступне визначення вебсайту навчального закладу: «…сукупність електронних документів, які висвітлюють достовірну інформацію про нормативні засади й основні напрями діяльності ВНЗ, об'єднаних однією електронною адресою, доменним ім'ям або IP–адресою [36]».

Головко О.А. виділяє чотири основні функції вебсайту вищого навчального закладу [37]:

– реклама та позиціонування освітніх послуг. Вебсайт повинен мотивувати абітурієнтів обрати саме наш ВНЗ, що є особливо важливим в період жорсткої конкуренції між численними навчальними закладами;

– створення та підтримка іміджу ВНЗ. Це охоплює такі процеси, як позиціонування ВНЗ, ініціювання відповідних комунікаційних процесів та контроль над ними, створення образу та представлення про навчальний заклад у мережі;

– комунікація з потенційними клієнтами (вступниками). Це передбачає процес отримання зворотнього зв'язку від потенційних вступників, аналіз успішності залучення абітурієнтів, моніторинг інтересів цільової аудиторії;

– зв'язок з суспільством. Це передбачає розміщення публікації про ВНЗ, які демонструватимуть його гарні сторони, освітні послуги, різноманітні презентації, участь в наукових заходах, віртуальні екскурсії по закладу освіти.

Загалом, структуру типового вебсайту закладу освіти можна поділити на наступні складові частини:

– історія ВНЗ, або структурного підрозділу, його місії та перспективи;

– управлінська структура ВНЗ. Ректорат, деканати, факультети, кафедри, викладацький колектив;

– новини та анонси;

– розклад занять;

– блок, посвячений науковій діяльності. Участі в конференціях, семінари, наукові досягнення студентів та викладачів, збірники тез, фахові журнали;

– інформація пов'язана з вступною компанією (перелік іспитів, терміни);

– студентське життя, гуртожитки, масові заходи;

– міжнародна діяльність ВНЗ;

– кар'єрні можливості;

– електрона бібліотека/репозитарій;

– форми для зворотнього зв'язку;

– опитування та анкетування.

Діяльність структурних підрозділів регулюється [38] та звучить наступним чином: Структурні підрозділи утворюються рішенням вченої ради закладу вищої освіти у порядку, визначеному цим Законом і статутом закладу вищої освіти. Основними структурними підрозділами закладів вищої освіти (крім коледжів, які не здійснюють підготовку бакалаврів) є факультети, кафедри, бібліотека. Також сюди відносяться навчально–дослідні лабораторії.

Сайт структурного підрозділу повинен обов'язково мати посилання на головний сайт, а також свою історію створення та місію. Такий сайт повинен всебічно висвітлювати напрями роботи, демонструвати наукові досягнення його учасників, містити останні новини та події, та різними способами зацікавлювати відвідувачів.

Дизайн такого сайту повинен бути ергономічний та враховувати визначальні потреби користувачів. Критичну роль у забезпеченні задоволення користувачів відіграють: чітка структура навігації, мінімалістичний та адаптивний дизайн, інтерактивні елементи взаємодії, підтримка високоякісного мультимедійного контенту. Кольорова палітра та графічний дизайн повинні відображати корпоративний стиль ВНЗ. Максимально недоречними на такому сайті будуть рекламні банери (якщо це не пов'язано з діяльністю ВНЗ).

Для прикладу було виділено основні критерії аналізу, та проведено порівняння 2–ох різних вебсайтів структурних підрозділів закладів освіти. В якості першого сайту було взято вебсайт Навчально–наукового Інституту телекомунікаційних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (рисунок 2.1).

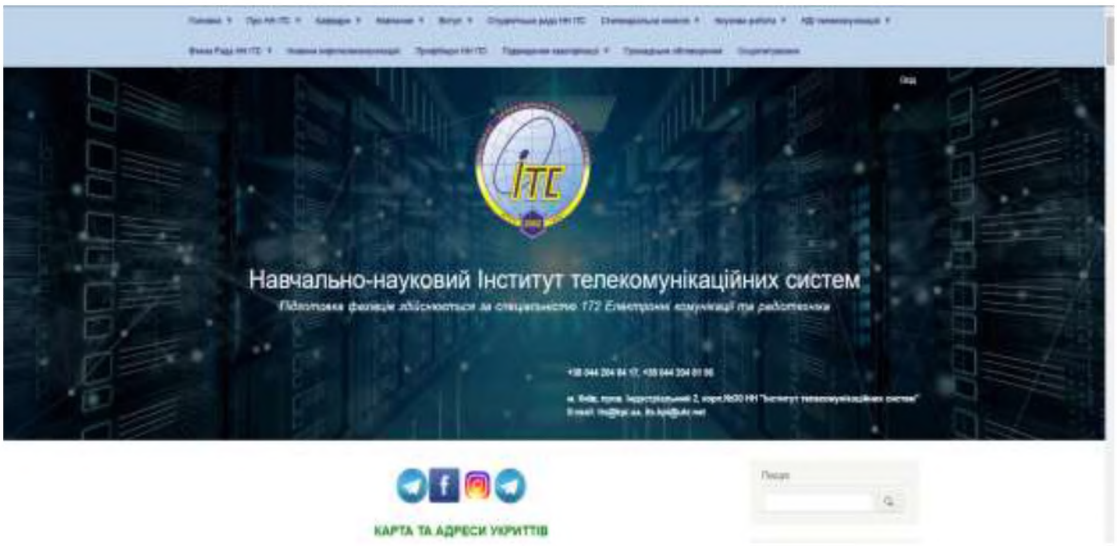


Рисунок 2.1 – Головна сторінка Навчально–наукового Інституту телекомунікаційних систем [39]

В якості другого сайту взято Науково–дослідний STEM–центр Тернопільського національного педагогічного університету імені Володимира Гнатюка (рисунок 2.2).

Рисунок 2.2 – Головна сторінка Науково–дослідного STEM–центру [40]

Для порівняння спеціально було обрано сайти, які значно відрізняються як наповненням і призначенням, так і дизайном. Сайт на рисунку 2.1 виконано в більш діловому, класичному стилі. В той же час, сайт на рисунку 2.2 виконано в більш вільному та сучасному стилі.

На основі порівняння було сформовано таблицю 2.1 В цій таблиці кожному критерію надається оцінка від 0 до 5 – чим вище бал, тим краще реалізований той чи інший функціонал.

Таблиця 2.1 – Порівняння вебсайтів структурних підрозділів закладів освіти

Критерій	Навчально–науковий інститут телекомунікаційних систем	Науково–дослідний STEM–центр
Наявність системи пошуку на сайті	5	2
Наявність інших мов інтерфейсу	3	0
Історія та місія підрозділу	5	5
Наукові досягнення учасників	4	5
Новини та події	5	5
Можливість зворотнього зв'язку	4	4
Зручність навігації по сторінкам сайту	4	5
Посилання на основний сайт ВНЗ	1	1
Інформація про викладачів	3	1
Інтеграція з соціальними мережами	5	0
Інтуїтивність	3	4
Швидкодія	2	5
Адаптивність для мобільних пристроїв	5	5
Середній бал	3,76	3,23

З результатів аналізу обраних вебсайтів (див. таблицю 2.1) можна зробити висновки, що не всі ВНЗ надають достатньої уваги вебсайтам своїх структурних підрозділів, наприклад, обидва вебсайти не мають очевидного посилання на головний сайт ВНЗ. В той же час на обох вебсайтах наявна можливість зворотнього зв'язку, добре висвічуються новини та події, а також наявна зручна навігація між сторінками. Обидва сайти мають чудову версію для мобільних користувачів. При створенні проєктної частини кваліфікаційної роботи будуть враховані переваги та недоліки, виявлені під час цього аналізу.

Вебсайт структурного підрозділу призначений для того, щоб користувачі могли з легкістю дізнатись про сьогодення Навчально–дослідної лабораторії вебтехнологій та хмарних обчислень.

Даний вебдодаток повинен мати наступні можливості:

– авторизація;

– адміністративна панель;

– методи зворотнього зв’язку;

– пошук статей по назві;

– пошук робіт по назві;

– лічильник кількості перегляду статті;

– посилання на головний вебсайт ПДАУ;

– історія та інформація про викладачів лабораторії.

Сайт повинен складатися з 2 частин: адміністративної та користувацької. Адміністратор повинен мати змогу додавати, редагувати та видаляти статті та роботи студентів.

Інтерфейс користувача вебсайту повинен бути інтуїтивно зрозумілим, логічно представляти структуру розміщеної на ньому інформації та мати можливість легкого та швидкого переходу між розділами додатку.

На сайті планується реалізувати 4 користувацькі та 1 адміністраторську сторінку, а також можливість створення необмеженої кількості новин. Окрім цього планується створення модального вікна для зворотнього зв’язку.

Перелік сторінок:

– Головна. Тут будуть розміщені останні новини лабораторії, корисні посилання та будуть представлені найкращі роботи студентів;

– Про лабораторію. Тут повинна розміщуватись інформація про працівників лабораторії та її останні новини;

– Роботи студентів. Тут будуть розміщені роботи студентів в рамках діяльності лабораторії. Планується зберігати лише основну інформацію про роботу студента – автора, короткий опис, та посилання на зовнішній хостинг. Планується, що студенти самі будуть виставляти свої роботи на окремий

хостинг. Це буде добрим тренінгом для них, а також зменшить навантаження на додаток;

– Новини. Тут будуть розміщуватись новини структурного підрозділу;

– Окрема сторінка новини. Для кожної новини буде автоматично створюватися окрема сторінка, зроблена по шаблону;

– Адмін–панель. Сторінка, доступ до якої буде тільки в авторизованих користувачів. Тут можна буде відслідковувати кількість статей, робіт студентів, переглядів, та керувати контентом, що розміщений на сайті.

Окрім цього планується створення допоміжних сторінок:

– Сторінка авторизаці. Сюди перенаправлються користувачі, які намагаються виконати дію, що вимагає авторизації;

– Сторінка помилки «404».

Структурна схема сайту наведена в додатку В. Для реалізації поставлених задач необхідно обрати інструментарій розробника.

2.2 Порівняльні характеристики та вибір інструментарію проєктування вебдодатку

Для виконання практичної частини роботи спершу необхідно визначитися з інструментарієм.

Перед початком реалізації вебсайту потрібно створити макет майбутнього сайту. Макет – це візуальне концепція, що передує розробці самого сайту. Завдяки макету можна до найменших подробиць продумати, як буде виглядати майбутній продукт: розташування елементів, кольорова гамма, шрифти та загальна структура сторінок.

Для створення макетів можна використовувати один з наступних інструментів: Adobe Photoshop, Figma, Zeplin, Framer, Sketch. Для порівняння було створено таблицю 2.2. Кожному критерію проставлена оцінка від 1 до 5, після чого порахована сума балів.

Таблиця 2.2 – Порівняння графічних редакторів

Критерій	Adobe Photoshop	Figma	Zeplin	Framer	Sketch
Легкість використання	2	5	4	2	4
Командна розробка	3	5	4	2	3
Інтеграція з іншими інструментами	5	4	4	3	4
Прототипування	2	5	1	4	3
Швидкість роботи	3	5	3	4	4
Доступність	3	5	4	4	4
Функціональність для дизайну інтерфейсів	5	5	4	4	4
Екосистема плагінів	4	4	3	3	4
Всього	27	38	27	26	30

За результатами порівняння (див. таблицю 2.2), найкращим варіантом виявився інструмент Figma. Це хмарна технологія для створення макетів, в тому числі вебсайтів. Однією з ключових особливостей Figma є можливість спільної роботи в реальному часі, що дозволяє дизайнерам, розробникам та іншим учасникам команди одночасно працювати над проєктом з різних пристроїв. Інтерфейс Figma зображено на рисунку 2.3.

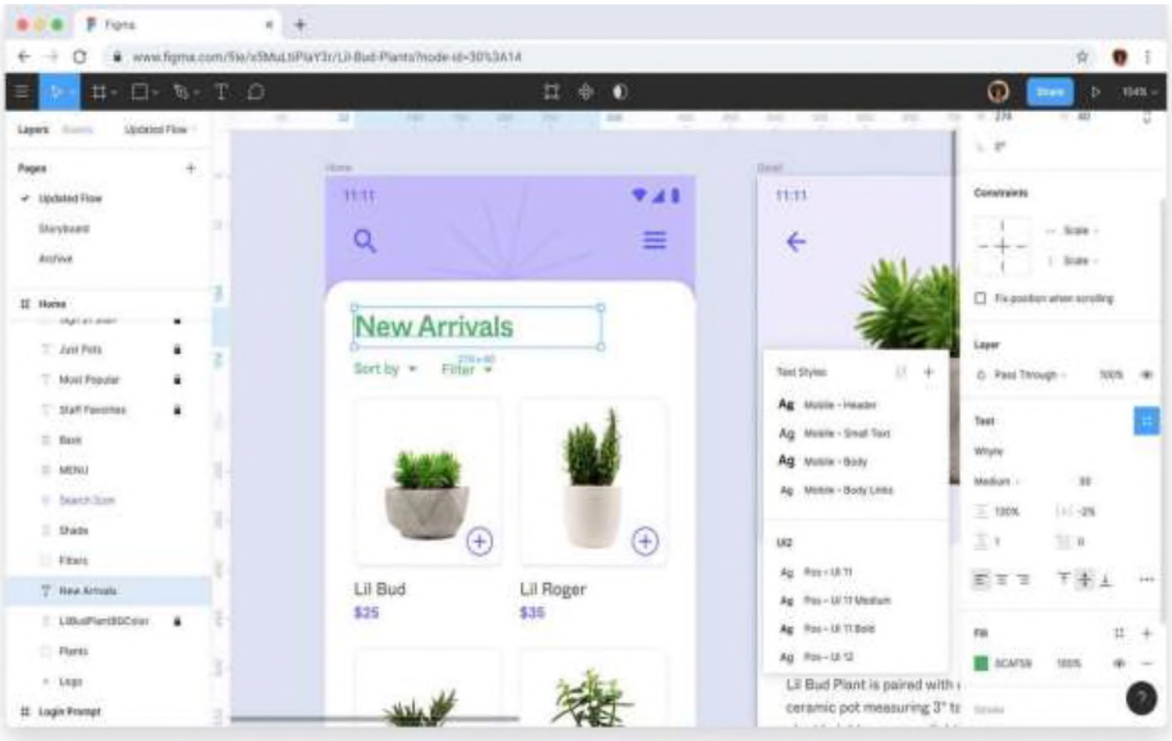


Рисунок 2.3 – Інтерфейс Figma

В попередньому розділі було встановлено, що для розробки frontend частини бажано використовувати фреймворк. Ми вже знаємо, що найпопулярнішими фреймворками є: React, Angular, Vue та Svelte. Для вибору найкращого з них сформуємо таблицю переваг та недоліків (таблиця 2.3).

Таблиця 2.3 – Переваги та недоліки популярних JavaScript фреймворків

Фреймворк	Переваги	Недоліки
React	Висока швидкодія; Компонентний підхід; Одностороннє зв'язування даних; Велика спільнота та екосистема; Можливість використання JSX	Високий поріг входження; Великий розмір бібліотеки; Проблеми із SEO–оптимізацією; Напружена взаємодія з реальним DOM в окремих випадках
Angular	Обов'язкове використання TypeScript; Великі можливості для тестування; Декларативний синтаксис шаблонів; Система залежностей та ін'єкції залежностей	Обов'язкове використання TypeScript; Великий розмір бандлів; Складність вивчення; Обмежена гнучкість; Нища швидкодія ніж в інших фреймворках
Vue	Легкість вивчення; Гнучкість; Зручний синтаксис шаблонів; Можливості реактивного програмування; Малий розмір	Менша спільнота порівняно з React та Angular; Менша кількість готових рішень; Брак стандарту в іменуванні; Ризик надмірної гнучкості.
Svelte	Компіляція на етапі збору; Легкість вивчення; Істинна реактивність; Відсутність Virtual DOM; Вбудований препроцесор для стилів.	Мала спільнота; Мала кількість готових рішень; Обмежена інтеграція в екосистему; Можливі питання з масштабованістю.

Аналіз особливостей фреймворків React, Angular, Vue та Svelte, а також їх переваг та недоліків (див. таблицю 2.3) показав, що універсальним рішенням для будь–якого проєкту є React. Найкращим фреймворком для великих проєктів є Angular. Для стартапів та невеликих проєктів гарним вибором буде Vue. Якщо ж вам потрібна продуктивність, то варто зупинитись на Svelte [41].

Враховуючи всі особливості обираємо для нашого вебдодатку React.

React – це бібліотека (не фреймворк!) з відкритим кодом, створена компанією Facebook, яка використовується для створення користувацьких інтерфейсів, а саме для SPA (Single–page Application). За допомогою React розробники можуть з легкістю створювати об'ємні вебдодатки, які здатні

оновлювати інформацію на сторінці, без оновлення самої сторінки. Це реалізовано за допомогою Virtual DOM. VirtualDOM це концепція, в якій ідеальне, або «віртуальне» відображення UI зберігається в пам'яті і синхронізується з реальним за допомогою бібліотек–посередників. Коли в React інтерфейсі відбувається зміна, то відбувається rerendering лише той частини DOM–дерева, яка зазнала змін. Головною особливістю React є використання JSX. JSX – це розширення синтаксису JavaScript. Цей механізм дозволяє писати HTML структури всередині JavaScript файлу використовуючи синтаксис JSX. На відміну від інших фреймворків, в яких зазвичай реалізується two–way data binding (як наприклад в Angular), в React реалізований однонаправлений потік даних. Це дає більший контроль над даними, що подорожують всередині вебдодатку. React має гарну зворотню сумісність. Дуже малоймовірно, що після оновлення версії фреймворку в проєкті щось перестане працювати. React можна використовувати для вебдодатків будь–якого розміру: як маленьких застосунків по типу калькулятора, або гри в змійку, так і для масштабних проєктів як от Facebook.

Окрім React, на frontend'і нам знадобляться ще деякі бібліотеки:

– Redux Toolkit – офіційна бібліотека від команди Redux для керування станом (стейтом) вебдодатків. Вона включає в себе такі корисні функції як: configureStore – для створення сховища (місце для зберігання даних), createSlice –для створення редюсерів та екшенів (для взаємодії зі стейтом), а також createAsyncThunk – для виконання асинхронних операцій (взаємодіїї з сервером). Його використання дозволить зберігати всю інформацію про стан додатку в одному місці та легко ним керувати;

– React–router – бібліотека для керування маршрутизацією в React–додатках. За допомогою неї можна керувати адресним рядком, та змінювати компоненти без перезавантаження сторінки. За допомогою таких компонентів як «Route» та «Link» визначаються шляхи та навігаційні елементи додатку. Використання цієї бібліотеки значно спрощує розробку додатків середнього та великого розміру;

– EditorJS – специфічна бібліотека, потрібна саме в нашому випадку. Вона являє собою готовий WYSIWYG (What You See Is What You Get) редактор тексту. Особливістю саме цього редактору є те, що він не формує відразу готовий HTML–код, а зберігає дані в форматі JSON, що дозволяє розробнику за необхідності виконувати різні дії з ними. Він знадобиться нам для створення та редагування статей;

– Axios – бібліотека для спрощення роботи з HTTP–запитами. В Axios реалізовані інтерцептори(interceptors), використання яких дозволяє встановлювати заголовки, перехвачувати та обробляти помилки, та автоматично перетворювати дані в формат JSON. Він є потужнішим аналогом стандартного fetch.

Для вибору фреймворку для backend'у порівняємо популярні фреймворки: Spring, Django, Express.js, Laravel. Для вибору найкращого з них сформуємо таблицю переваг та недоліків кожного з них (таблиця 2.4).

Таблиця 2.4 – Переваги та недоліки популярних backend фреймворків

Фреймворк	Переваги	Недоліки
Spring	Широкий набір модулів і функціоналу; Inversion of Control, Dependancy Injection, аспектно–орієнтоване програмування, модульність, багата екосистема	Комплексініть; навантаження на пам'ять; складність навчання; складність конфігурації; велика кількість абстракцій
Django	Орієнтований на конвенції конфігураційний підхід; автоматизована адміністративна панель; загальна стандартизація; високий рівень безпеки	Невисока гнучкість у виборі компонентів; високі апаратні вимоги; не повністю асинхроний, великі URL–шляхи
Express.js	Легкість вивчення; гнучкість та мінімалізм; активна спільнота та екосистема; підтримка middleware; швидкість та продуктивність	Брак стандартизації; callback hell; малий функціонал «з коробки»; відсутність вбудованої підтримки ORM
Laravel	Чистий синтаксис; модульність; використання ORM Eloquent; система маршрутизації; система шаблонів Blade, безпека	Велика ресурсомісткість; навантаження на сервер, відносна складність навчання; залежність від Composer

Враховуючи результати порівняння переваг та недоліків (див таблицю 2.4) для використання було обрано Express.js.

Express – серверний фреймворк для створення вебдодатків на основі Node.js. Заснований на концепції MVC, він доповнює Node високорівневим інтерфейсом для обробки HTTP–запитів та функціоналом middleware. Middleware дозволяє обробляти запити перед тим, як вони потрапляють до обробника маршруту. Роутинг Express.js дозволяє чітко визначати, як обробляються різні запити, а обробники маршруту визначають функції для конкретних маршрутів. Окрім цього, Express.js надає функціонал для роботи з різними движками шаблонізації та обробки статичних файлів. Загалом, Express є чудовим фреймворком для розробки серверної частини нескладних вебзастосунків.

Окрім самого фреймворку нам знадобляться наступні бібліотеки:

Mongoose – спеціальна ODM (Object Data Modelling) бібліотека для роботи з MongoDB. Вона пропонує високорівневу абстракцію у вигляді структурного інтерфейсу для визначення моделей даних та операцій з БД;

Bcrypt – бібліотека для хешування паролів за допомогою адаптивної криптографічної хешфункції формування ключа, засновані на шифрі Blowfish.

Multer – бібліотека, що надає middleware функціонал для роботи з файлами. Вона потрібна нам для обробки та зберігання графічних рисунків;

Cloudinary – сервіс/бібліотека, яка буде використана нами в якості сховища файлів;

Cors – бібліотека, що надає middleware для роботи з cors (Cross–origin resource sharing);

Cookie–parser – бібліотека, що дозволяє аналізувати заголовок Cookies і працювати з об'єктом req.cookies. Це нам знадобиться під час розробки системи авторизації та автентифікації;

Jsonwebtoken – бібліотека для створення та верифікації JWT (JSON Web Tokens).

В якості бази даних було обрано NoSQL базу – MongoDB.

MongoDB – це документоорієнтована система керування базами даних (NoSQL), особливістю якої є представлення даних у вигляді BSON–

документів (бінарний JSON). У MongoDB дані організовані у вигляді колекцій, що складаються з документів. Документ – це структурований об'єкт у форматі BSON, який може включати в себе різнорідні дані, включаючи вкладені масиви та документи. Відсутність строгої схеми дозволяє динамічно змінювати структуру даних в межах однієї колекції. Завдяки горизонатльному масштабуванню дані можна розподіляти по декільком серверам.

Ця БД чудово підходить для роботи в парі з Express.js. Express вміє працювати з асинхронними операціями, що поєднується з такою ж асинхронною MongoDB. Окрім цього, в нашій роботі нам не знадобиться встановлювати складні зв'язки між даними в БД, отже NoSQL є кращим рішенням ніж SQL база. В якості СУБД для MongoDB буде використано MongoDBCompass (рисунок 2.4).

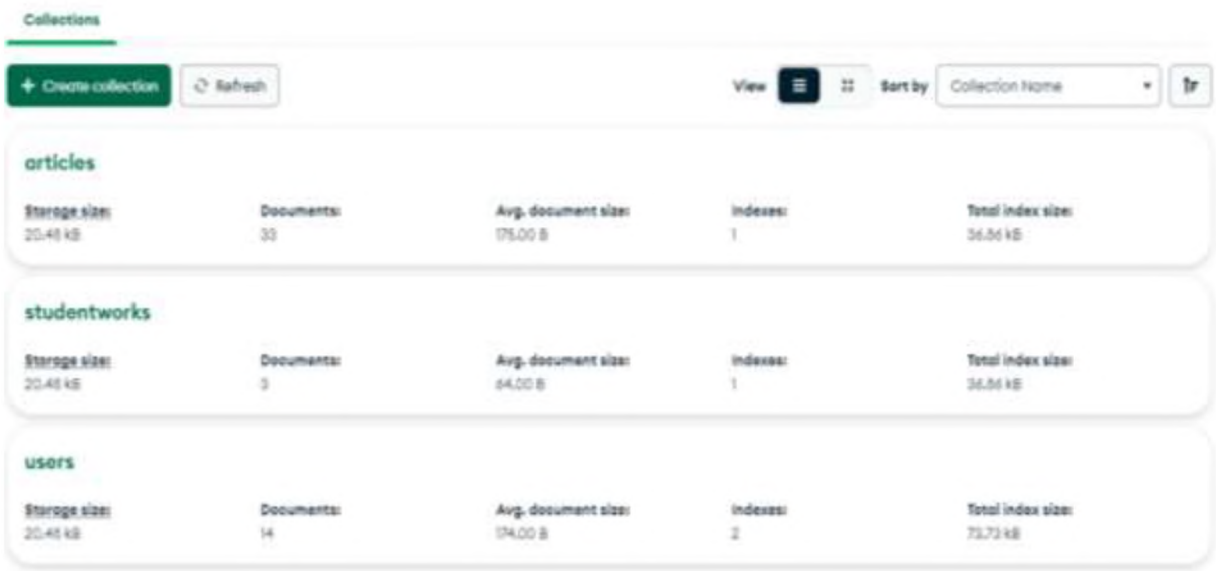


Рисунок 2.4 – Інтерфейс роботи з колекціями MongoDBCompass

Сукупність обраного інструментарію, являє собою сучасний стек технологій, названий MERN (Mongo, Express, React, Node). Він дозволяє розробникам створювати повнофункціональні вебзастосунки від начала й до кінця, використовуючи JavaScript як на клієнтській, так і на серверній частині. Це зменшує складність створення додатку та спрощує передачу даних між клієнтом та сервером. Схема роботи MERN Stack зображена на рисунку 2.5.

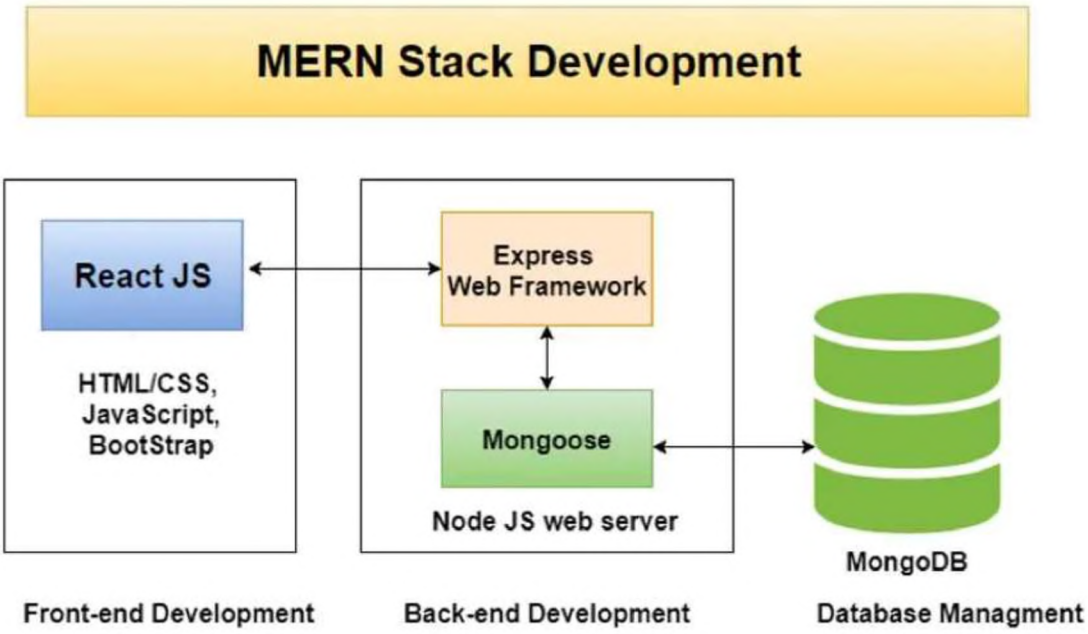


Рисунок 2.5 – Схема роботи MERN Stack [42]

Як сказано вище, MERN Stack складається з 4–ох технологій та реалізує паттерн MVC. MongoDB відіграє роль бази даних, в той час як Node.js та Express використовуються для розробки API (backend), а React для реалізації користувацького інтерфейсу (frontend).

Наступним є середовище розробки. Потрібно провести порівняльний аналіз наявних рішень та обрати найбільш підходящий варіант. Оскільки для реалізації frontend частини було обрано React, а для backend частини – Node.js, будемо порівнювати наступні інструменти: Sublime Text, Visual Studio Code, WebStorm, Atom, Text Mate. Результати порівняння занесено в таблицю 2.5.

Таблиця 2.5 – Порівняння середовищ для розробки

Критерій	Sublime Text	Visual Studio Code	WebStorm	Atom	Notepad++
Зручність інтерфейсу	+	+	+	+/–	+/–
Підтримка мов програмування	+	+	+	+	–
Розширення та плагіни	+	+	+	+	+
Відлагодження (Debugging)	–	+	+	+	–
Спільна робота (Collaboration)	–	+	+	+	–
Інтеграція з системами керування версіями	–	+	+	+	–
Швидкодія та продуктивність	+	+	–	+	+
Наявність безкоштовної версії	+	+	+/–	+	+

В результаті порівняння (див. таблицю 2.5) очевидним фаворитом виявився Visual Studio Code. Розглянемо його більш детально.

Visual Studio Code – це легкий, але потужний редактор коду, доступний для Windows, Linux та MacOS. В ньому є вбудована підтримка JavaScript, TypeScript та Node.js. Редактор також містить вбудовані інструменти для навігації по коду, автодоповнення базових мовних конструкцій та контекстні підказки. Однією з головних переваг VSCode є екосистема плагінів [43].

Для виконання завдання нам знадобляться наступні плагіни:

– ES7+ React/Redux/React Native – плагін, що спрощує роботу з React шляхом надання сніппетів та підтримки ES7 синтаксису;

– Prettier – плагін для форматування коду. Дозволяє з легкістю писати код в одному стилі, підтримує JSX синтаксис;

Зовнішній вигляд VSCode представлено в Додатку Г.

Перейдемо до пакетного менеджеру. Нам він обов'язково потрібен, оскільки ми будемо працювати з великою кількістю бібліотек, як на фронтенді так і на бекенді. Потенційними варіантами є npm та Yarn. Потрібно провести їх порівняльний аналіз. Результати порівняння занесено в таблицю 2.6.

Таблиця 2.6 – Порівняння пакетних менеджерів

Критерій	Yarn	npm
Швидкість	Швидкий	Повільний
Підтримка оффлайн	Вбудована	Потребує додаткової конфігурації
Безпека	Сканує пакети на вразливості	Використовує npm audit д
Багатопоточність	Операції виконуються паралельно, що пришвидшує встановлення пакетів	Послідовне виконання операцій (повільніше)
Автоматичне скорочення	Автоматично видаляє непотрібні залежності	Потрібує ручного введеня "npm prune"
Популярність	Лише набирає популярність	Дуже популярний та широковикористовуємий
Продуктивність	Швидко працює з великою кількістю залежностей	Відносно повільний

Як бачимо (див. таблицю 2.6), обидва, Yarn та npm мають свої переваги та недоліки. В той час, як npm є «старожилом» для JavaScript та має велику

спільноту та репозитарій, Yarn надає більшу швидкість, захищеність та можливості для оффлайн–розробки.

Враховуючи плюси та мінуси обох менеджерів, для використання під час розробки кваліфікаційної роботи було обрано Yarn.

Серед систем контролю версії безперечним лідером є Git. Це open–source інструмент, який дозволяє ефективно відстежувати і контролювати зміни в проєктів. До особливостей Git можна віднести:

– збереження абсолютно всієї історію проєкту у вигляді commit'ів. В будь–який час можна повернути стан проєкту до попереднього;

– гілки, що являють собою окремі незалежні лінії розробки, та дозволяють розробникам паралельно працювати над різними частинами проєкту;

– підтримка інтеграції з такими сервісами як GitHub, GitLab, Bitbucket.

Окрім системи контролю, потрібен ще онлайн–репозитарій. Це місце, де розробники можуть спільно працювати над програмним кодом. Git та зовнішні репозитарії наразі є нероздільними інструментами.

Хоча над вебдодатком працюю я сам, використання таких інструментів все рівно є гарною практикою, оскільки це дозволить мені зберігати всі файли в хмарі, і у випадку форс–мажору, проєкт не прийдеться переробляти з нуля. Найпопулярнішими репозитаріями є: GitHub, GitLab, Bitbucket. Для порівняння створимо таблицю 2.7.

Таблиця 2.7 – Порівняльні характеристики репозитаріїв

Критерій	GitHub	GitLab	Bitbucket
Репозитарій для коду	+	+	+
Соціальна взаємодія	+	+/–	–
DevOps	+/–	+	+/–
Інструменти проєктного менеджменту	+	+	+
Баг–трекінг	+	+	+
Асистент – штучний інтелект	+	–	–
Інтеграції	+	+	+/–
Безпека	+	+	+
Підтримка open–source проєктів (безкоштовно)	+	+	–
Автоматизація за допомогою CI/CD	+	+	+

В результаті аналізу (див. таблицю 2.7) найкращим варіантом виявився GitHub. Його і будемо використовувати.

В якості системи автоматизації збірки було обрано Webpack.

2.3 Окремі методології розробки програмних продуктів та спеціальне програмне забезпечення підтримки проєктної діяльності

Важливість управління проєктами в бізнес–компаніях, організаціях важко переоцінити. Коли управління ведеться правильно, це допомагає кожній частині бізнесу працювати більш якісно. Це дозволяє команді зосередитися на важливій роботі, не відволікаючись від завдань, які збиваються з плану, або бюджетів, які виходять з–під контролю. Зрештою, це дає їм змогу досягати результатів, які фактично впливають на прибутки бізнесу. Співробітники мають можливість побачити, як їх робота сприяє досягненню стратегічних цілей компанії.

Існує велика кількість традиційних методів і підходів до розробки програмного забезпечення. Це такі моделі як водоспадний, ітераційний, тощо. Інша назва цих підходів – підходи до планової розробки програмного забезпечення. Такі підходи є дуже корисними, коли виникає потреба розробити велике комплексне ПЗ.

Водоспадна (каскадна) модель. Найстаріший і найвідоміший представник. В ній кожний етап розробки продовжує попередній. Тобто, щоб перейти на наступний етап, потрібно повністю завершити роботу над поточним. В умовах постійних змін вимог, використання такої моделі може перетворитися з переваги на недолік. Зараз каскадною моделлю в основному користуються великі компанії при роботі над великими і складними проєктами, де важливий всеосяжний контроль ризиків.

Перевагами цієї моделі є:

– повна документація;

– чітке планування витрат та термінів;

– прозорість процесів.

Недоліки:

– необхідність затвердження всього плану ще на першому етапі, в разі зміни вимог можлива переробка всього проєкту [44].

Ітераційна модель. В такій моделі життєвий цикл продукту розбивається на велику кількість міні–циклів (ітерацій), кожна з яких відповідає за розробку окремого компонента системи, після чого цей компонент додається до загальної системи. Ітеративна модель складається з чотирьох основних етапів (plan–do–check–act): визначення та аналіз вимог, дизайн та проєктування, розробка і тестування, рев'ю.

Переваги ітераційної моделі:

– рання реалізація працюючої версії ПЗ;

– гнучкість;

– кожна ітерація є окремим етапом, тестування яких є простішим ніж тестування всього проєкту.

Недоліки:

– можуть виникнути проблеми з реалізацією загальної архітектури системи, оскільки не всі вимоги відомі до початку проєктування;

– неясність обсягу робіт [44].

Популярним сучасним підходом є використання принципів Agile. Це концептуальна структура розробки ПЗ, що починається с фази планування та триває до фази деплою, поступово та ітеративно взаємодіючи протягом всього життєвого циклу. Мета гнучких методологій полягає в зменшенні накладних витрат, що виникають в процесі розробки ПЗ, та забезпеченні можливості впровадження змін без ризику для процесу.

Однією з реалізацій принципів Agile є Scrum – це методологія управління проєктами, особливо популярна у розробці програмного забезпечення. В її основі лежить концепція спринтів. Спринт – це певний період часу (2–4 тижня), за який команда розробників повинна створити

придатний до випуску продукт. Під час кожного спринту команда збирає з беклогу продукту список вимог, які вони планують виконати протягом цього періоду. Після цього створюється план по виконанню задач з беклогу.

Вибором робочих задач для спринту займаються product–owner (власник продукту) та scrum–майстер і команда розробників. Scrum–майстер – лідер, який сприяє ефективному впроваджению та дотриманню Scrum–процесів в команді розробників. Наявність компетентного scrum–майстра допомагає команді пристосуватись до змін максимально ефективно та безболісно.

Під час спринту прововодяться Scrum–наради, на яких на обговорення виноситься хід роботи та виявляються основні проблеми та блокери. Результати роботи демонструються на огляді результатів спринту. Тут можна продемуонстру

Після спринту проводиться ретроспектива, під час якої команда аналізує результати спринту.

Для Agile–методологій характерним є використання Канбан–дошок (рисунок 2.6).

Рисунок 2.6 – Канбан–дошка

Канбан–дошка – це інструмент управління процесами в проєкті, що базується на візуальному відображенні завдань та їх статусів за допомогою дошки з завданнями [45]. Мета полягає в оптимізації потоку роботи, покращенні комунікації та ефективному розподілі завдань серед команди. Зазвичай Канбан–дошка поділяється на 3 області – зробити, в роботі, готово. Але інколи команди самостійно додають інші області, наприклад: на паузі (on hold) або відкладено/призупинено (paused). Завдяки цьому команда може легко відстежувати прогрес, відслідковувати проблемні задачі, та грамотно використовувати ресурси.

Для інформаційної підтримки проєктної діяльності в сучасному світі використовується спеціальне програмне забезпечення – системи управління проєктами. Основна мета цих систем – забезпечити ефективне планування, виконання та контроль проєктів.

Згідно аналітичних досліджень, найпопулярнішими системами управління проєктами у 2023 р. є Jira і Microsoft Project. Менш популярними є: Asana, Trello, Smartsheet (рисунок 2.7).

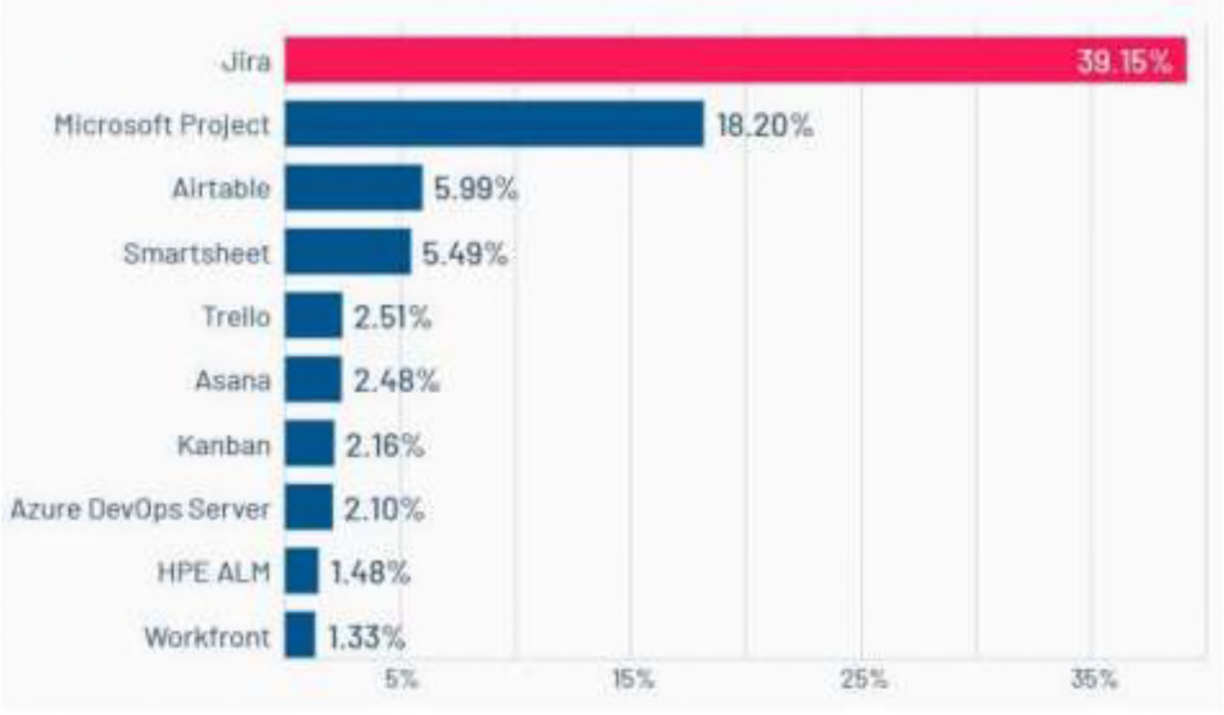


Рисунок 2.7 – Статистика використання систем управління проєктами станом на 2023 р. [46]

Порівняння популярних інструментів приведено в таблиці 2.8.

Таблиця 2.8 – Порівняльний аналіз особливостей функціональних можливостей та складності використання популярних систем управління проєктами

Назва СУП	Особливості функціональних можливостей	Складність використання
Jira	Окрім стандартних можливостей, Jira також має додатково: створення backlog, розробка roadmap, розподілення і коментування задач, аналіз продуктивності співробітників, легко розширюється і впроваджується з другими системами	Jira доволі складна СУП через велику кількість вбудованих функцій
Microsoft Project	Створення та управління зв'язками між задачами, управління ресурсами, можливість встановлення віх та критичних задач, управління вартістю та бюджетом, формування різних звітів, діаграма Ганта, налаштування різних календарів і варіантів представлення проєктів. Наявна інтеграція з продуктами Microsoft. Це найбільш комплексна СУП із представлених	MS Project може бути дещо складним для людей, які раніше не працювали з подібним ПЗ, через велику кількість функцій
Asana	Виділяється системою структурування проєкту шляхом розбиття на папки та розділи. Можлива інтеграція з іншими сервісами (Dropbox, Google Drive, Adobe Creative Cloud). Із мінусів – можливість призначення лише одного виконавця задачі, відсутність планування ресурсів, діаграми Ганта	Asana має зручний інтерфейс та просту систему навігації, що дозволяє швидко освоювати її
Trello	Створення дошок для проєктів та розподілення карток за списками (lists) відповідно до етапів виконання, створення карток з описом завдань та відповідальними за їх виконання користувачами. Відсутня діаграма Ганта за замовчування (необхідно встановлювати спеціальний плагін)	Trello має дружній до користувача інтерфейс і буде зрозумілим навіть для людей, які раніше не працювали в системі управління проєктами

Виходячи з результатів аналізу (див. таблицю 2.8), можна зробити висновок, що кожна з розглянутих систем має свої переваги та недоліки. Наприклад для великих проєктів краще підійде Microsoft Project, або Jira, які мають потужніший функціонал. Для менших проєктів краще використовувати Asana, або Trello, оскільки вони є менш функціональними, але в той же час простішими в використанні [47].

Оскільки вибір системи управління проєктами залежить від потреб конкретного проєкту та команди, важливо попередньо ретельно проаналізувати можливості та вимоги, щоб визначити, яка система буде найбільш ефективною для його управління.

Висновки до розділу 2

В цьому розділі нами було встановлено, що являє собою вебсайт закладу освіти, та конкретно вебсайт структурного підрозділу. Проаналізовано вимоги до таких вебсайтів та проведено порівняння існуючих аналогів, для виявлення основних моментів, на які варто звернути увагу під час виконання практичної частини роботи.

Сформульовано та поділено на логічні кроки задачу практичної частини кваліфікаційної роботи. Вона полягатиме в створенні сучасного вебдодатку для Навчально–дослідної лабораторії вебтехнологій та хмарних обчислень. Поставлено вимоги до дизайну, розроблено майбутню структуру та вимоги до функціоналу вебдодатку.

Було проаналізовано, обрано та описано різноманітний інструментарій для реалізації: фреймворки, бібліотеки, базу даних, систему контролю версій та репозитарій, пакетний менеджер, інтегроване середовище розробки, систему автоматичної збірки та графічний редактор. Створено велику кількість порівняльних таблиць, обгрунтовано вибір кожного інструменту.

Окрім цього було розглянуто різні сучасні методології та підходи до розробки програмних продуктів. Проведено порівняльний аналіз функціональних можливостей систем управління проєктами.

РОЗДІЛ 3
ОПИС ПРАКТИЧНОЇ ЧАСТИНИ РЕАЛІЗАЦІЇ ВЕБДОДАТКУ

3.1 Програмна реалізація backend частини вебдодатку

Для початку потрібно створити серверну (backend) частину. В попередньому розділі ми обрали Node.js + Express + MongoDB для цього.

Спершу треба створити базу даних. Для цього потрібно перейти на офіційний сайт MongoDB [48] та зареєструватися. Після цього потрібно створити кластер, та базу. Коли кластер буде створено, нам запропонують підключити свій Node.js проєкт. Для цього нам нададуть спеціальний рядок, який треба буде вставити в наш код (рисунок 3.1). Зараз ми можемо підключитись до бази даних в MongoDBCompass використовуючи ці дані. Але цей рядок нам ще знадобиться пізніше, тож зберігаємо його.

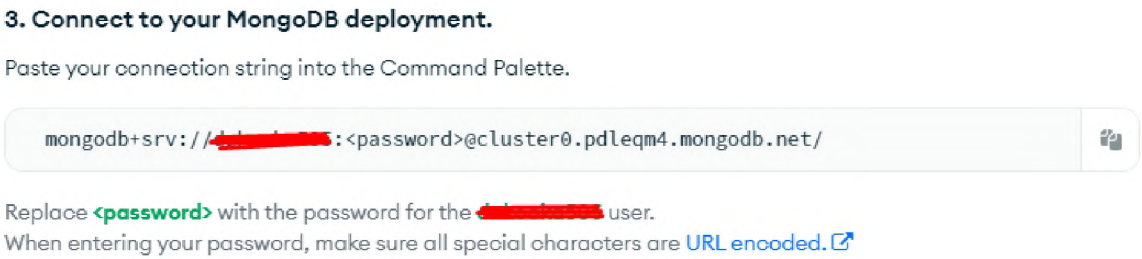


Рисунок 3.1 – Підключення до MongoDB

MongoDB вже зараз проявляє свої особливості – нам не треба створювати ніяки таблиці та документи. Вони будуть створені автоматично, при виконанні перших запитів.

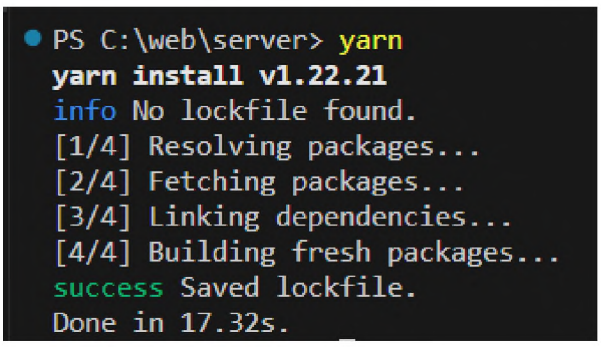
Оскільки весь програмний код є дуже великим навіть для розміщення в додатках, ознайомитись з повним варіантом можна в репозитарії GitHub [49]. В даному розділі будуть наведені лише фрагменти коду, які наглядно щось демонструють (тобто без повторень за функціоналом і лише найважливіші).

Для створення Node.js проєкту нам знадобляться вже розглянуті в попередньому розділі бібліотеки (рисунок 3.2).

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "dev": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "cloudinary": "^1.41.0",
    "cookie-parser": "^1.4.6",
    "cors": "^2.8.5",
    "dotenv": "^10.0.0",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.2",
    "mongodb": "^6.2.0",
    "mongoose": "^8.0.0",
    "multer": "^1.4.5-lts.1",
    "uuid": "^9.0.1"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

Рисунок 3.2 – Перелік бібліотек для backend–частини

Завантажуємо всі залежності за допомогою команди yarn (рисунок 3.3). Всі завантажені файли будуть зберігатись в папці ./node_modules, яку потрібно додати в спеціальний файл ./.gitignore, щоб вона не завантажувалась на GitHub.



```
PS C:\web\server> yarn
yarn install v1.22.21
info No lockfile found.
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...
success Saved lockfile.
Done in 17.32s.
```

Рисунок 3.3 – Встановлення необхідних бібілотек за допомогою Yarn

Структура папок проєкту виглядає наступним чином:

– controllers/ – папка для файлів–контролерів, в яких буде міститись основна логіка отримання–передачі даних та взаємодії з БД.

– dtos/ – папка для Dto (Data Transfer Object) файлів;

– exceptions/ – універсальний обробник помилок;

– middlewares/ – функції–посередники, наприклад перевірка авторизації, або обробка непередбачених помилок;

– models/ – файли з описом схем об'єктів, що будуть зберігатися в БД;

– utils/ – ок кові функції, наприклад для роботи з токенами, або зі стороніми сервісами;

– router/ – опис кінцевих точок, та включення функцій посередників;

– інші допоміжні файли;

Для реалізації нам потрібні наступні кінцеві точки (таблиця 3.1).

Таблиця 3.1 – Кінцеві точки проєкту

Шлях	Тип	Призначення
/api/registration	POST	Для реєстрації нових користувачів, в нашому випадку, для додавання нових адміністраторських аккаунтів
/api/login	POST	Для авторизації в додатку
/api/logout	POST	Для виходу з аккаунту
/api/refresh	GET	Для оновлення токену доступу
/api/getStatistics	GET	Для отримання статистики (кількість робіт, новин, переглядів)
/api/addArticle	POST	Для додавання нової статті
/api/getSingleArticle	GET	Для отримання одної статті (по id)
/api/getAllArticles	GET	Для одержання всіх статей
/api/updateArticle	PUT	Для оновлення існуючої статті (по id)
/api/deleteArticle	DELETE	Для видалення однієї статті (по id)
/api/addWork	POST	Для додавання нової роботи
/api/getSingleWork	GET	Для отримання одної роботи (по id)
/api/getAllWorks	GET	Для одержання всіх робіт
/api/updateWork	PUT	Для оновлення існуючої роботи (по id)
/api/updateWork	DELETE	Для видалення однієї роботи (по id)
/api/uploadFile	POST	Для завантаження файлів (зображень в новинах) на сервіс Cloudinary
/api/addCarouselItem	POST	Для додавання елементу каруселі
/api/getAllCarouselItems	GET	Для отримання всіх елементів каруселів
/api/getSingleCarouselItem	GET	Для отриманні одного елементу каруселі (по id)
/api/updateCarouselItem	PUT	Для оновлення одного елементу каруселі (по id)

Для запуску проєкту використовується команда yarn run dev. В головному файлі index.js встановлюються всі основні налаштування: використання cookieParser та cors, вказуються кінцеві точки та middlewares. Після чого сервер намагається підключитись до MongoDB, та в разі успіху сервер запускається на порту 5000.

Потрібно створити схеми. Це спосіб описання структури документа в колекції, шляхом встановлення полів та типів даних, валідаторів та інших опцій. Потім, на основі цих схем створюються моделі, за допомогою яких можна взаємодіяти з базою даних. В нашому проєкті нам необхідно створити 4 схеми – UserSchema, ArticleSchema, StudentWorkSchema, CarouselSchema (таблиця 3.2).

Таблиця 3.2 – Схеми, створенні в додатку

Схема	Поле	Вимоги	Призначення
UserSchema	email	String, unique, required	Електрона пошта адміністратора
	password	String, required	Пароль від аккаунта
	refreshToken	String	Рефреш–токен доступу
ArticleSchema	title	String, required, unique	Заголовок (назва) статті
	content	String, required	Вміст статті у вигляді рядка з HTML–тегами
	originalJSON	Mixed	Спеціальний JSON–об'єкт, який сприємається редактором EditorJS
	author	User	Автор статті
	date	Date	Час створення статті у форматі (число–місяць–рік)
	viewCounter	Number	Лічильник переглядів статті
StudentWorkSchema	url	String, required	Посилання на роботу студента
	title	String, required	Заголовок (назва) роботи
	backgroundImage	String, required	Фонове зображення роботи на сайті (прев'ю або тематична картинка)
	author	String	Автор роботи
	date	Date	Час створення роботи у форматі (число–місяць–рік)
CarouselSchema	url	String,required	Посиланеня на елемент
	title	String,required	Заголовок
	description	String,required	Текстовий опис
	image	String,required	Посилання на зображення

В нашому проєкті ми використовуємо концепцію DTO для уніфікації та структурування данних. Приклад такого класу наведено на рисунку 3.4. В конструкторі такого класу задаються всі поля об'єкта. За бажанням можна змінювати їх назву. Наприклад, MongoDB автоматично створює всім об'єктам поле _id, яке є не дуже зручним. За допомогою DTO ми змінюємо його на просто id – без підкреслювання. Це допомагає уникнути путанини.

```
module.exports = class ArticleDto {
    id;
    title;
    content;
    originalJSON;
    author;
    date;
    viewCounter;

    constructor(model) {
        this.id = model._id;
        this.title = model.title;
        this.content = model.content;
        this.originalJSON = model.originalJSON;
        this.author = model.author;
        this.date = model.date;
        this.viewCounter = model.viewCounter
    }
}
```

Рисунок 3.4 – DTO–класс ArticleDTO

Для прикладу наведено реалізацію обробника кінцевої точки /api/login, який знаходиться в файлі user–controller.js (Додаток Д). Дана функція приймає логін та пароль від користувача, та перервіряє їх на валідність, після чого генерує пару токенів, записує їх в cookies, та повертає користувачу об'єкт, що містить інформацію про користувача, та новостворений токен доступу.

Приклад програмного коду схеми статті. За замовченням експортується екземпляр класу Schema, що міститься в бібліотеці Mongoose. Саме його ми будемо використовувати в других функціях для роботи з БД (рисунок 3.5).

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

const ArticleSchema = new Schema({
    title: { type: String, required: true, unique:true },
    content: { type: String, required: true },
    originalJSON: { type: Schema.Types.Mixed },
    author: { type: Schema.Types.ObjectId, ref: 'User' },
    date: { type: Date },
    viewCounter: {type: Number}
})

module.exports = model('Article', ArticleSchema);
```

Рисунок 3.5 – Вміст файлу article–model.js

Для прикладу візьмемо функцію отримання однієї статті (рисунок 3.6). Ми беремо id з query запиту, та шукаємо по ньому статтю за допомогою функції findById(), якщо її нема то інформуємо про це користувача. Якщо стаття існує, то додаємо їй один перегляд та відправляємо користувачу.

```
async getSingleArticle(req, res, next) {
    try {
        const { id } = req.query; // дізнаємося id статті
        const article = await ArticleModel.findById(id).populate('author', '_id, email');
        if (article) {
            if (article.viewCounter) {
                article.viewCounter++; // додаємо перегляд
            }
            await article.save()
            return res.json(article); // повертаємо статтю
        }

        return res.json({ message: 'Статті не існує' })
    } catch (e) {
        next(e) // обробка помилок
    }
}
```

Рисунок 3.6 – Функція getSingleArticle в файлі article–controller.js

Важливою складовою додатку є авторизація. В нашому проєкті вона буде реалізована за допомогою JWT–токенів. Це такий стандарт відкритих токенів, що передає інформацію в форматі об’єкту JSON. Вони

зашифровуються спеціальним алгоритмом, приймаючи як ключ кодове слово (в нашому випадку ключові слова записані в файл .env зміні JWT_ACCESS_TOKEN та JWT_REFRESH_TOKEN). Їх потрібно надійно зберігати, оскільки саме за допомогою них відбувається верифікація токенів.

Існує 2 види токенів – refresh (токен оновлення) та access (токен доступу). Токени оновлення використовуються для отримання нової пари токенів, при закінченні терміну дії попередніх, а токени доступу – для отримання доступу до закритих кінцвеих точок. Токен оновлення повинен передаватись виключно в HttpOnly–cookies. Таким чином блокується доступ JavaScript та забезпечується захист від деяких видів вразливостей, наприклад від XSS (Cross Site Scripting).

У випадку, якщо зловмисник спробує підробити інформацію в будь–якій з частин токену, то зфальсифікований токен виявиться не валідним, оскільки його підпис не буде відповідати первинному значенню. Окрім цього, хакер не зможе створити новий підпис, бо в нього немає доступу до секретного ключа, який зберігається на сервері.

В нашому проєкті тривалість життя токенів буде: 15 хвилин для access та 30 днів для refresh (рисунок 3.7). Токен доступу буде зберігатись в Local Storage, а токен оновлення – в cookies. Токен складається з 3 частин:

– заголовок (Header) – містить 2 поля (тип токену та алгоритм підпису);

– корисна інформація (Payload) – будь–яка інформація, яку ви хочете зашифрувати і передати в токені;

– підпис (Signature) – результат кодування заголовка та корисної інформації за допомогою алгоритму підпису, вказаного в заголовку. Підпис додається до токена для забезпечення його цілісності та перевірки автентичності при отриманні.

Найпопулярнішим алгоритомом для підпису токену є HS256. Також допускається використання ассиметричного алгоритму RS256. Його відміністю є те, що він створє два ключа: публічний та приватний. Приватний використовується для створення підпису, а публічний лише для перевірки.

```
const generateTokens = (payload) => {
    const accessToken = jwt.sign(payload, process.env.JWT_ACCESS_TOKEN, { expiresIn: '15m' }) //
    const refreshToken = jwt.sign(payload, process.env.JWT_REFRESH_TOKEN, { expiresIn: '30d' })
    return {
        accessToken,
        refreshToken
    }
}
```

Рисунок 3.7 – Приклад функціоналу токенів

Окрім контролерів та обробки токенів, важливою частиною коду є middlewares. Наприклад middleware для обробки помилок (рисунок 3.8) потрібен, щоб перехвачувати всі помилки, які розробник не оброблює вручну. Це важливо, оскільки без обробки помилок, це б приводило до постійного падіння серверу.

```
const ApiError = require('../exceptions/api-error');

module.exports = function (err, req, res, next) {
    console.log(err);
    if (err instanceof ApiError) {
        return res.status(err.status).json({message: err.message, errors: err.errors})
    }
    return res.status(500).json({message: 'Непередбачувана помилка!'})

};
```

Рисунок 3.8 – Вміст файлу error–middleware.js

Для перевірки правильності (і взагалі!) роботи серверу використовується раніше розглянутий програмний засіб Postman. Наприклад, після створення функціоналу для додавання статті необхідно його перевірити. Для цього в Postman створюємо нову кінцеву точку (в нашому випадку http://localhost:5000/api/addArticle), встановлюємо в Headers токен авторизації (оскільки доступ на цей ендопінт повинні мати тільки авторизовані користувачі), та формуємо тіло запиту. Після чого, в разі успіху, ми отримаємо відповідь від сервера. Для наглядності всі ці кроки послідовно продемонстровано на рисунку 3.9.

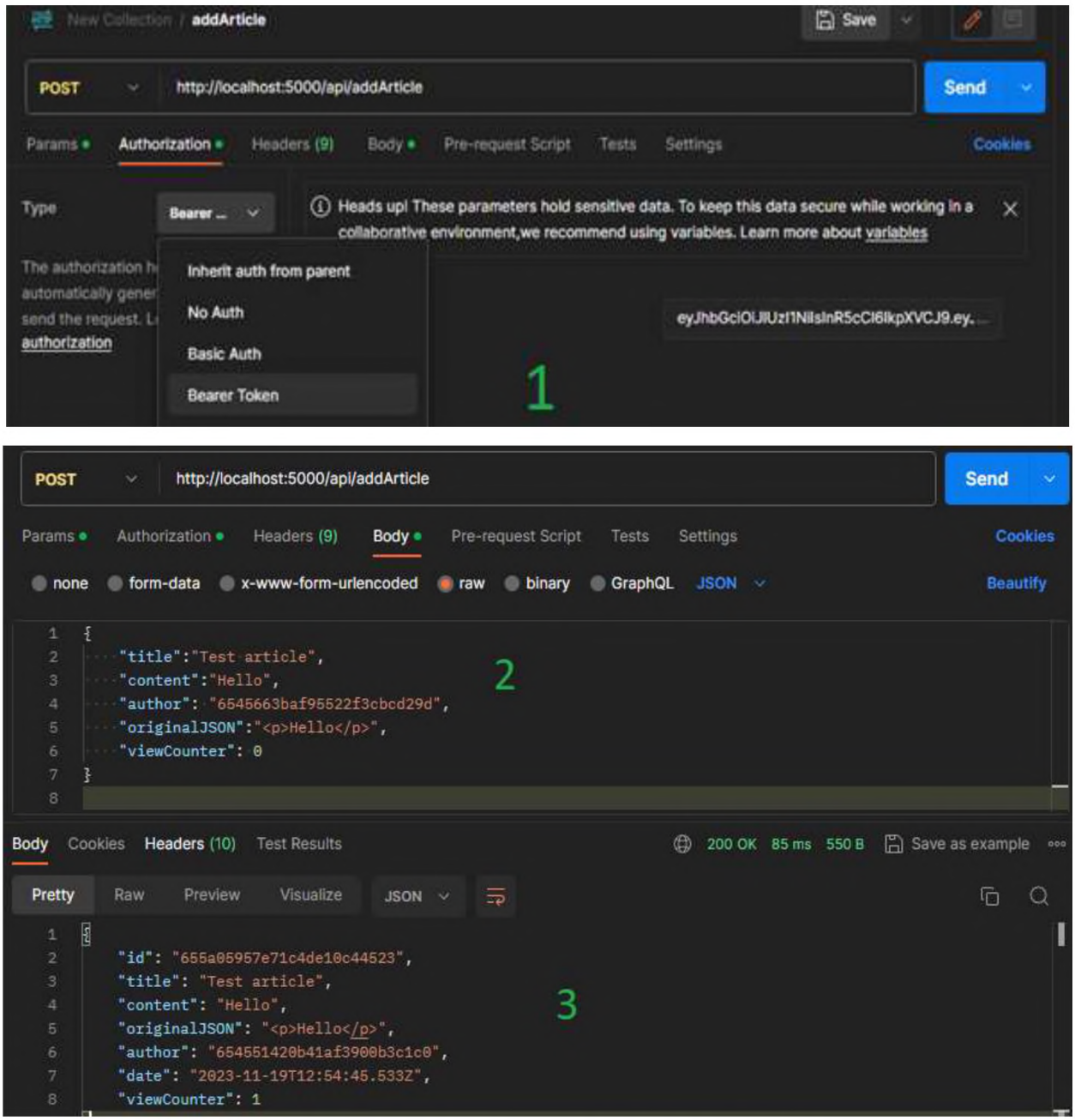


Рисунок 3.9 – Приклад роботи з Postman

Після цього ми можемо побачити в нашій БД новий запис, який відповідає відправленій раніше інформації (рисунок 3.10). Варто відзначити, що колекція articles створилась автоматично, нам не потрібно було її створювати вручну – лише описати модель всередині нашого коду. Після чого MongoDB автоматично створює колекції під час нашого запиту. Всього в нашому проєкті 3 колекції, по одній на кожну модель. В MongoDB за необхідності можна вручну видаляти записи в колецекції, або колекцію повністю.

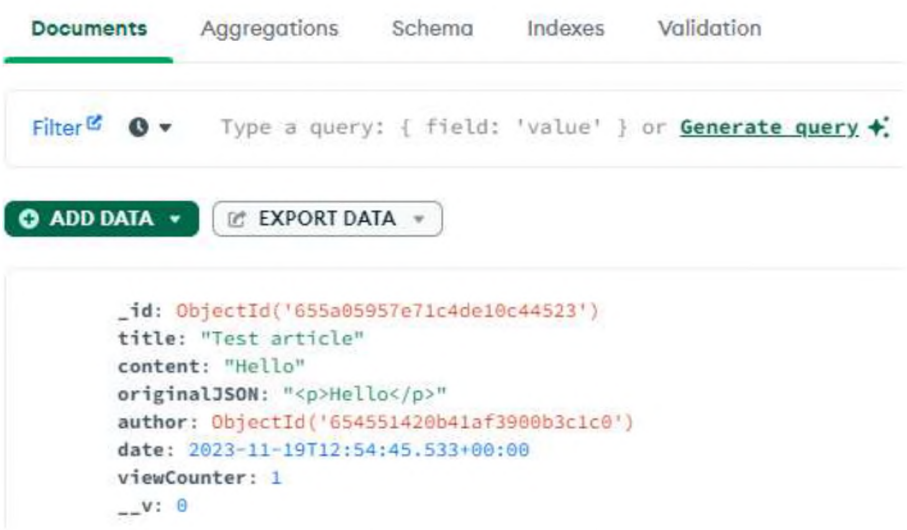


Рисунок 3.10 – Приклад документу MongoDB

Для завантаження коду на репозитарій GitHub та встановлення на хостинг робимо наступні дії:

1. Створюємо на GitHub новий публічний репозитарій (рисунок 3.11).

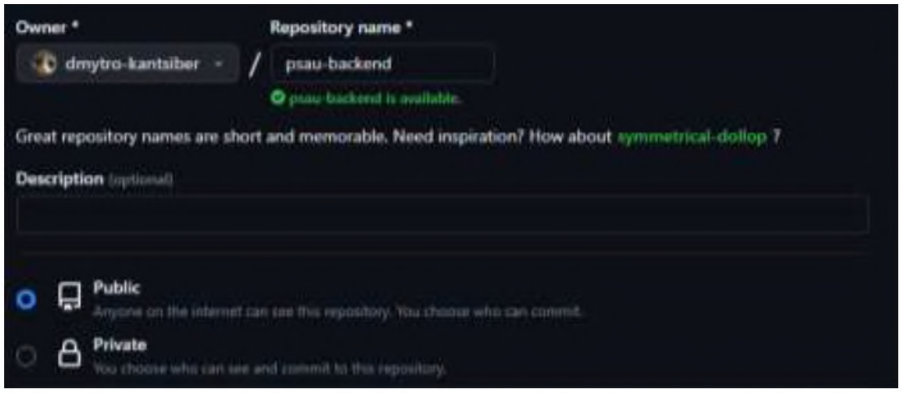


Рисунок 3.11 – Створення нового репозитарію на GitHub

2. Виконуємо послідовно всі команди, необхідні для завантаження файлів (рисунок 3.12).

```
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/dmytro-kantsiber/psau-backend.git
git push -u origin main
```

Рисунок 3.12 – Процес завантаження проєкту на GitHub

3. В результаті бачимо весь наш код на GitHub (рисунок 3.13)

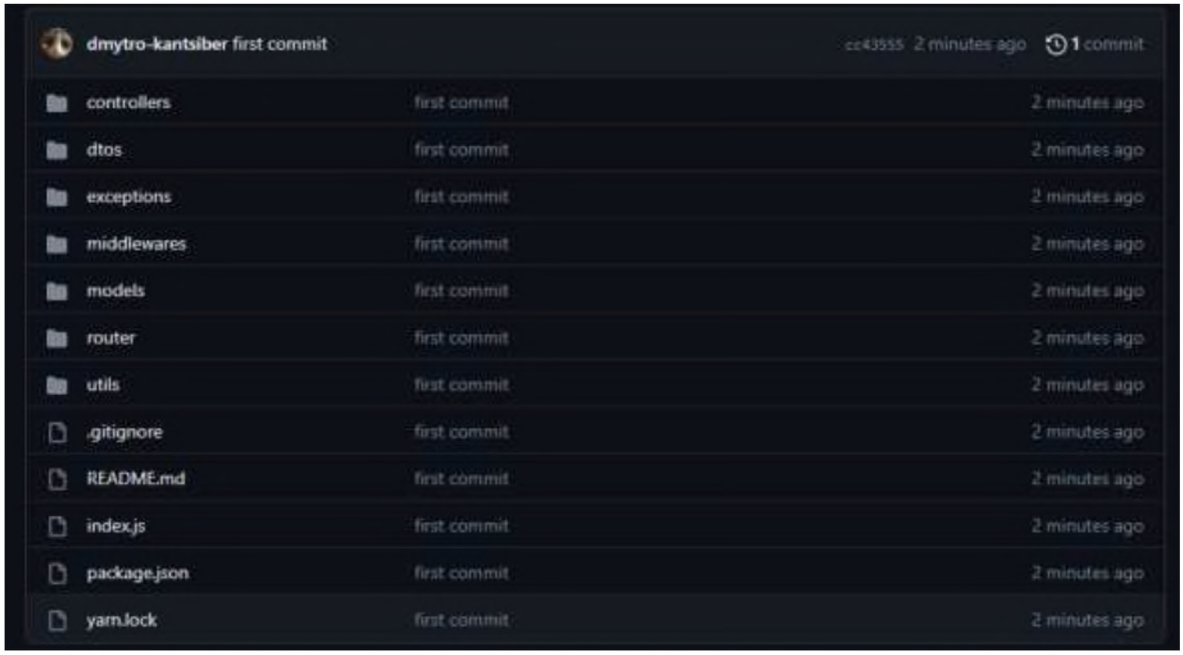


Рисунок 3.13 – Репозитарій backend частини додатку

4. Для хостингу будемо використовувати безкоштовний сервіс dashboard.render. Мінусом цього хостингу є те, що при відсутності запитів до серверу протягом 15 хвилин, він входить в режим сну, а після цього йому треба декілька хвилин, щоб знову запрацювати. Це тимчасове рішення, в подальшому планується перенесення серверу на повноцінний платний хостинг, де він буде доступний без затримок.

Хостинг на цій платформі є макисмально зручним, якщо ми використовуємо GitHub. Сервіс одразу пропонує нам підключитись до існуючого репозитарію (рисунок 3.14).

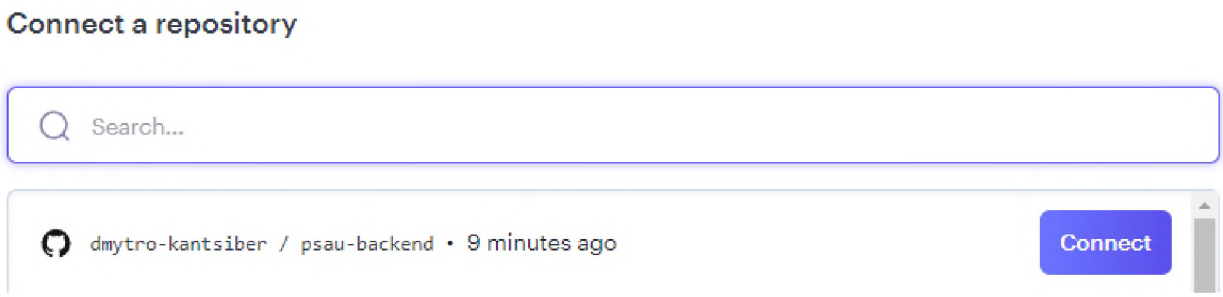


Рисунок 3.14 – Підключення до існуючого репозитарію

5. Після цього ми вводимо наші environment–зміні, налаштовуємо команди збірки та запуску проєкту (yarn та yarn dev run) та чекаємо результат. В результаті в нашому профілі з'явиться відповідний сервіс, та до нього можна буде звертатись із запитами по створеним кінцевим точкам (рисунок 3.15).

Рисунок 3.15 – Перевірка роботи сайту

В результаті роботи над backend частиною була створена та протестована система авторизації, CRUD (create–read–update–delete) контролери для кожної моделі, та middleware для обробки будь–яких помилок. З готовою серверною частиною можна приступати до розробки користувацького інтерфейсу.

3.2 Програмна реалізація frontend частини вебдодатку

Робота над фронтендом починається зі створення макетів сторінок майбутнього вебдодатку. В попередньому розділі ми обрали Figma в якості графічного редактора. Дизайн сайту буде виконано в мінімалістичному стилі, з одним головним кольором. На кожній сторінці буде однаковий header та footer. В хедері буде реалізована навігація по сайту, а також перехід на основний вебсайт університету, а в футері буде службова інформація та перехід до авторизації. Макет головної сторінки наведено на додатку Е.

Базовий React–проєкт створюється за допомогою команди npx create–react–app <назва_вашого_проєкту> (рисунок 3.16). Створений проєкт містить

всю необхідну базову конфігурацію, оптимізовану структуру та налаштовані параметри для Babel та Webpack. Звичайно, його можна створити і вручну, але це абсолютно не має сенсу, оскільки займе набагато більше часу та не матиме ніяких переваг порівняно з готовим рішенням.

Рисунок 3.16 – Результат виконання create–react–app

Структура папок проєкту буде наступною:

– /src/assets/images/ – статичні зображення, такі як логотип або фотографії керівників лабораторії;

– /src/assets/styles/ – папка для зберігання всіх файлів стилів (*.css);

– /src/components/common/ – папка для компонентів, що можуть повторюватися в різних частинах додатку (футер, хедер тощо);

– /src/components/pages/ – папка для зберігання компонентів, що будуть являти собою сторінки (тобто кожна міститиме окрему url–адресу);

– /src/utils/hooks/ – папка для кастомних хуків (розберемо що це пізніше);

– /src/utils/slices/ – папка для зберігання «слайсів». Це частина бібліотеки redux–toolkit, яку ми розглянемо пізніше;

– /src/utils/ – окрім згаданих вище папок, в даній папці будуть зберігатися наступні файли: api.js – для налаштування axios інтерсепторів, data.js – для зберігання статичних даних, store.js – для роботи з redux–toolkit, tools.js – для роботи з Editor.js;

– /src/App.js та index.js – служать точкою входу в додаток;

– /public/* – службові файли.

Аналогічно з бекенд–проєктом, необхідно встановити всі залежності та бібліотеки. Їх перелік наведено на рисунку 3.17.

Рисунок 3.17 – Перелік бібліотек для frontend частини

Завантажуємо всі залежності за допомогою команди yarn. Не забуваємо додати ./node_modules в файл ./.gitignore.

Створюємо redux–сховище в файлі store.js (рисунок 3.18). Тут буде зберігатися state (стан) нашого додатку, наприклад: інформація про авторизацію, новини, роботи. Головною є функція configureStore, яка задає всі редюсери (обробники), які можна буде використовувати в проєкті для роботи зі сховищем.

```
export const store = configureStore({
    reducer: {
        auth: authReducer,
        articles: newsReducer,
        works: worksReducer,
    },
})
```

Рисунок 3.18 – Конфігурація redux–store

Кожен редюсер складається з екшенів, тобто дій. Для роботи з асинхроними запитами використовується вбудована в redux–toolkit функція createAsyncThunk(). На прикладі функції перевірки стану авторизації

(рисунок 3.19). Для різних варіантів відповіді серверу (fulfilled, rejected) ми встановлюємо відповідні редюсери.

```
export const checkAuthAsync = createAsyncThunk('auth/checkAuth', async (_, { rejectWithValue }) => {
  try {
    const response = await axios.get(`${API_URL}/refresh`, { withCredentials: true });
    localStorage.setItem('token', response.data.accessToken);
    return { user: { id: response.data.id, email: response.data.email } };
  } catch (e) {
    console.error("Помилка при перевірці авторизації:", e);
    return rejectWithValue(e.response?.data?.message);
  }
});
```

Рисунок 3.19 – Приклад асинхроного обробника в redux–toolkit

Для використання цих функцій в інших компонентах нам необхідно використовувати хук useDispatch, який надаватиме нам об'єкт диспетчера, якому можна буде передавати функції для виконання. Для отримання даних, що містяться в стейті, ми будемо використовуват інший хук – useSelector. Великою перевагою його використання є автоматичний ререндер компонентів, при зміні стану. В React ререндер компоненту можливий в 2 випадках: зміна стану в стейті, що пов'язаний з цим компонентом, або зміна батьківського компонента. Важливо це враховувати під час оптимізації коду.

Хуки – це функції, за допомогою яких можна «причепитись» до стану додатку і методів життєвого циклу компонентів React. Це сучасний підхід, який прийшов на зміну классовим компонентам. Найвживанішими хуками є useState (для додавання стейту в компонент) та useEffect (для обробки побічних ефектів). В нашому проєкті ми також створимо кастомний хук, який буде містити логіку, пов'язану з CSS класами (Додаток Ж).

Важливою частиною є конфігурація axios інтерсепторів (Додаток И). За допомогою них ми будемо встановлювати в заголовок запиту токен доступу, а також перехвачувати відповіді з серверу. Якщо ми отримаємо помилку 401, то буде створюватися нова пара токенів та повторно відправлятися оригінальний запит. Завдяки цьому, користувачу не треба буде авторизуватися кожні 15 хвилин (тривалість життя токена доступу), або після закриття–відкриття браузера.

Роутинг реалізовуємо за допомогою бібліотеки react–router v5. В головному файлі App.jsx встановлюються всі шляхи, які будуть в проєкті, та відповідні компоненти. Частина роутеру представлена на рисунку 3.20.

```
<Switch>
  <Route path="/about">
    <About />
  </Route>
  <Route exact path="/news">
    <Articles />
  </Route>
  <Route exact path="/news/create">
    <CreateArticle />
  </Route>
  <Route exact path="/news/:id">
    <SingleArticle />
  </Route>
  <Route path="/news/:id/edit">
    <EditArticle />
  </Route>
```

Рисунок 3.20 – Роутинг в нашому проєкті

Прикладом компонента, який використовується в нашому проєкті в багатьох місцях є Footer.jsx (Додаток К). В ньому наглядно видно використання JSX розмітки, redux–toolkit інструментарію, розглянутого раніше та робота з токенами.

В зв'язку з тим, що JSX–код займає доволі багато місця, він буде частково знаходитись в додатках. Весь інший код можна знайти аналогічно з серверним кодом – в репозитарії GitHub [50].

Для авторизації було створено окрему сторінку ./login. Користувачу пропонується ввести пошту та пароль (рисунок 3.21). В разі успішного вводу, користувач буде перенаправлений на спеціальну, закриту від звичайних користувачів сторінку – /admin. Якщо неавторизований користувач захоче потрапити на цю сторінку, то він буде перенаправлений на сторінку /login.

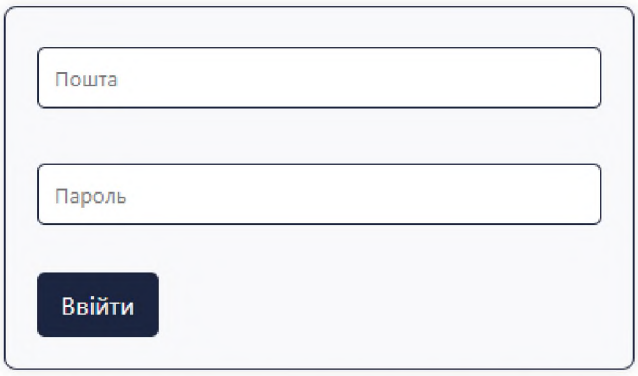


Рисунок 3.21 – Компонент авторизації – Login.jsx

В адміністративній панелі адміністратор може керувати статтями та роботами (додавати, редагувати, видаляти), створювати нові адміністраторські аккаунти та відслідковувати статистику (рисунок 3.22).

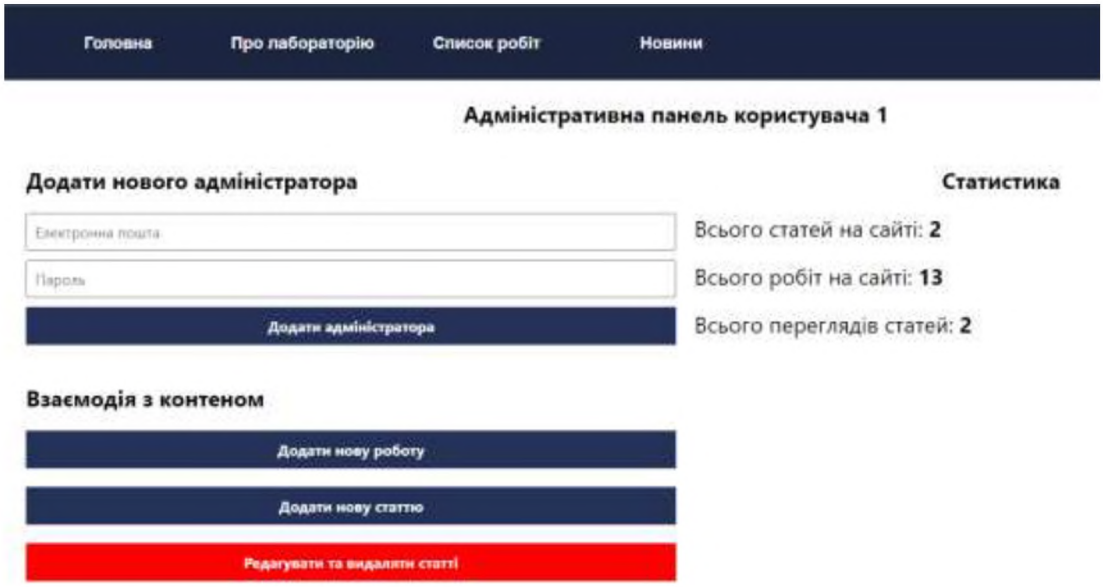


Рисунок 3.22 – Адміністративна панель – Admin.jsx

Для користувачів планується створити 4 доступних сторінки – Головна, Про лабораторію, Список робіт та Новини. Окрім цього, у кожної новини буде власна окрема сторінка за адресою /news/:id/. Кожну новину можна буде редагувати за посиланням /news/:id/edit. Сюди матиме доступ тільки авторизований користувач. Неавторизовані будуть перенаправлені на сторінку /login. Взагалі, з усіх сторінок, недоступних для звичайних користувачів, буде здійснюватися перенаправлення на сторінку авторизації.

На головній сторінці розміщується слайдер–карусель з найважливішою інформацією, яка може зацікавити нових користувачів. Нижче знаходяться 8 найкращих робіт студентів, та деякі корисні посилання по тематиці наукової діяльності лабораторії. Існує окрема сторінка /carousel, на якій авторизовані користувачі можуть змінювати наповнення слайдера–каруселі.

Сторінка «Новини». Програмний код сторінки розміщено в додатку Л. На цій сторінці знаходяться останні новини – від найновішої до найстарішої. Показується дата публікації. Реалізована пагінація з можливістю переходу на конретну сторінку, а також вперед–назад по одній. Максимум новин на сторінці – 8. Реалізований рядок пошуку, який буде шукати свівпадіння в назвах статтей і показувати лише відфільтровані результати. Для адміністраторів буде доступною кнопка «Видалати» на кожній новині. Вона, як зрозуміло з назви, видаляє статтю. Зовнішній вигляд сторінки наведено в додатку М.

Для того, щоб потрапити на сторінку статті потрібно натиснути будь–де в обведеній області. Кожна стаття містить інформацію про свого автора, дату створення (публікації) та кількість переглядів. Окрім того реалізована кнопка «Назад до новин», яка поверне користувача на сторінку всіх новин, враховуючи пошуковий запит та пагінацію. Якщо зайти на таку сторінку, будучи авторизованим, то буде доступна кнопка «Редагувати новину» (Додаток Н) яка при натиснені переправить вас на сторінку /news/:id/edit, де за допомогою компоненту Editor.jsx (Додаток П) можна зручно редагувати новину. За необхідності функціонал редактора можна змінювати в коді.

На сторінці «Про лабораторію» знаходиться основна інформація про діяльність, представлені її керівники з посиланням на їх сторінки на офіційному сайті ПДАУ.

Сторінка «Список робіт» дуже схожа на сторінку «Новини». Тут знаходяться всі роботи студентів. Так само реалізований пошук та пагінація. Наявна можливість редагувати та видаляти записи для авторизованих користувачів.

Окрім цього розроблена сторінка «404», яка буде відображатись на всих непередбачених адресах додатку.

Для всіх сторінок додатку розроблена мобільна версія (Додаток Р).

Розміщення коду на репозитарії GitHub відбувається ідентично до того, як ми це робили з бекендом. Едина відмінність – назва сховища. Для хостингу було прийнято рішення використати безкоштовний хостинг, який повністю інтегрований з GitHub – Vercel (рисунок 3.23)

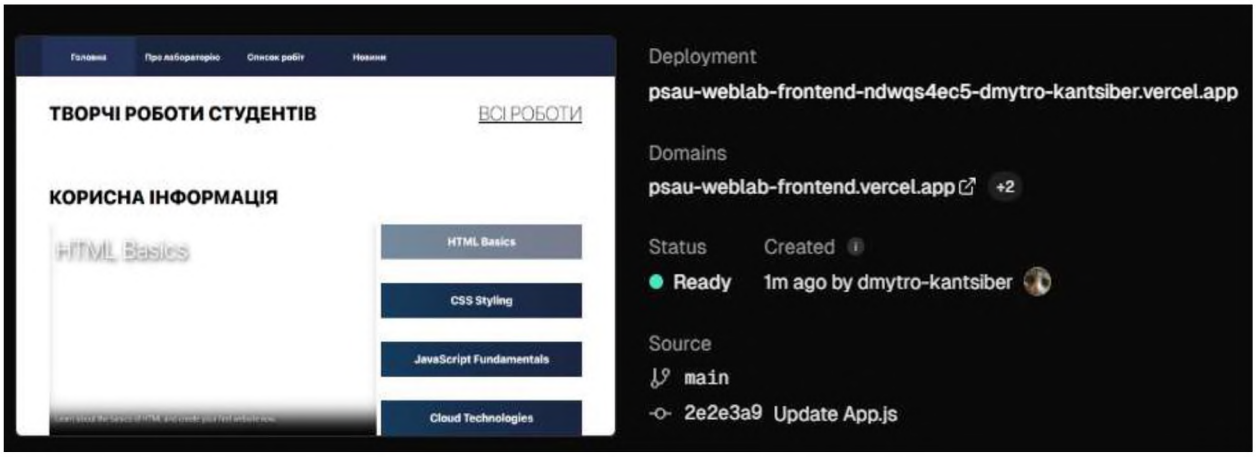


Рисунок 3.23 – Хостинг Vercel

В результаті роботи над frontend частиною було створено всі заплановані сторінки, та релізовано 100% запланованого функціоналу. Вихідний код проєкту розміщено на репозитарії GitHub та на хостингу Vercel.

3.3 Економічний аналіз розробки сучасного вебдодатку

В ході кваліфікаційної роботи було створено вебдодаток для структурного підрозділу закладу освіти. Такий сайт не буде приносити прямого прибутку, але буде сприяти загальному розвитку університету.

Враховуючи, що більшість сайтів все ж створюються з метою отримання прибутку, проведемо економічний аналіз, який включатиме: витрати на дизайн, розробку та різноманітні побічні витрати. Також порахуємо вартість

хостингу, оскільки безкоштовний хостинг не є гарним рішенням для розміщення сайту на постійній основі (особливо сайту нашого типу, оскільки він повинен містити доволі велику кількість інформації).

Економічний аналіз – це процес вивчення та оцінки економічних аспектів явищ, процесів, рішень чи проєктів з метою зрозуміння їхньої ефективності, вартості, можливих вигід чи ризиків. У контексті розробки вебдодатків це включає в себе:

– аналіз витрат;
– прогнозування прибутків;
– складання графіків та встановлення критеріїв;
– оцінку ринкового потенціалу;
– обгрунтування оптимальних стратегій.

Для обгрунтування доцільності розробки та впровадження проєктного рішення варто провести розрахунок наступних економічних показників:

– оплата праці;
– витрати на куповані вироби;
– транспортно–заготівельні витрати;
– накладні витрати;
– інші витрати.

Розмір оплати праці визначається за різними критеріями, такими як складність та умови виконуваної роботи, професійні якості працівника, вплив його трудової діяльності та економічної діяльності підприємства. Заробітна плата формується з основної та додаткової частин оплати праці.

Базова заробітня плата нараховується виходячи з тарифної сітки, договірних розцінок або посадових окладів, та не залежить від господарської діяльності підприємства.

Додаткова оплата праці – різноманітні матеріальні заохочення, наприклад – премія або надбавка за шкідливість. Нараховуються зазвичай по результатам запланованих та виконаних показників, кваліфікаційної підготовки, умов виробництва тощо.

Візьмемо для розрахунку мінімально необхідний склад команди розробинків: дизайнер (диз), програміст (прог) та тестувальник (тест). Візьмемо середню зарплату дизайнера – 20000 грн. на місяць, програміста – 30000 грн. на місяць та тестувальника – 15000 грн. на місяць.

Орієнтовно, розробка проєкту займе 2 тижня, тобто 10 робочих днів.

Працівники будуть працювати по стандартному 40–годиному графіку (8 годин на день). Виходячи з цього почасова оплата дизайнера – 114 грн, програміста – 170 грн, тестувальника – 85 грн.

Формула розрахунку заробітньої плати:

$$З_п = (К_{днів} \cdot В_{роб}) \cdot Т_г, де, \quad (3.1)$$

$К_{днів}$ – кількість робочих днів;

$В_{роб}$ – тривалість робочого дня в годинах;

$Т_г$ – годинна ставка розробника;

$$З_{диз} = (10 \cdot 8) \cdot 114 = 9120 \text{ грн.},$$

$$З_{прог} = (10 \cdot 8) \cdot 170 = 13600 \text{ грн.},$$

$$З_{тест} = (10 \cdot 8) \cdot 85 = 6800 \text{ грн.}$$

Витрати на придбання ($В_{пр}$) в нашому випадку становитимуть: USB–накопичувач – 200 грн., папір офісний (1 упаковка, 80 листів) – 90 грн., безлімітний інтернет на місяць – 250 грн. Всього 540 грн.

Визначення транспортно–заготівельних витрат. Транспортно–заготівельні витрати ($В_{тр}$) для розрахунків даного типу становлять 11% від суми витрат на придбання товарів.

$$В_{тр} = В_{пр} \cdot 0{,}11., де, \quad (3.2)$$

$В_{пр}$ – витрати на придбання;

$$В_{тр} = 540 \cdot 0{,}11 = 59{,}4 \text{ грн.}$$

Визначення накладних витрат. Наладні витрати ($В_н$) проєктних організацій складають 25% витрат на оплату праці.

$$В_н = (З_{диз} + З_{прог} + З_{тест}) \cdot 0{,}25, де, \quad (3.3)$$

$З_{диз}$ – зарплата дизайнера;

$З_{прог}$ – зарплата програміста;

$З_{тест}$ – зарплата тестувальника;

$$В_н = (9120 + 13600 + 6800) \cdot 0{,}25 = 7380 \text{ грн.}$$

Визначення інших витрат (електроенергія, вода тощо). Інші витрати ($В_{ін}$) передбачають всі витрати, що не враховані в попередніх розрахунках. Вони становлять 6% до витрат на оплату праці.

$$В_{ін} = (З_{диз} + З_{прог} + З_{тест}) \cdot 0{,}06, \text{ де,} \quad (3.4)$$

$З_{диз}$ – зарплата дизайнера;

$З_{прог}$ – зарплата програміста;

$З_{тест}$ – зарплата тестувальника;

$$В_{ін} = (9120 + 13600 + 6800) \cdot 0{,}06 = 1771{,}2 \text{ грн.}$$

Загальна сума витрат становить:

$$В_{сум} = (З_{диз} + З_{прог} + З_{тест}) + В_{тр} + В_н + В_{ін}, \quad (3.5)$$

$$В_{сум} = (9120 + 13600 + 6800) + 59{,}4 + 7380 + 1771{,}2 = 38730{,}6 \text{ грн.}$$

Окремо варто розглядати витрати на хостинг. Не кожен хостинг наразі підтримує розміщення Node.js додатків. Гарним рішенням з підтримкою необхідного інструментарію є платформа ukraine.com.ua (рисунок 3.24).

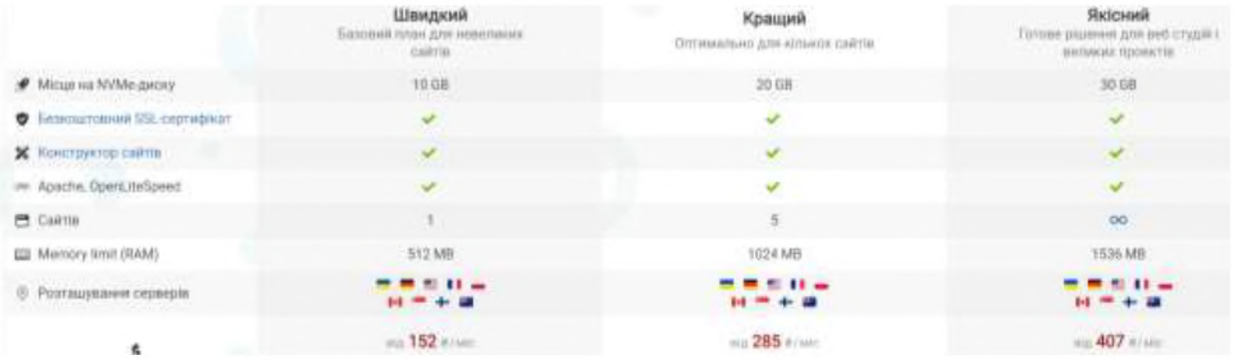


Рисунок 3.24 – Тарифи хостингу ukraine.com.ua

Найдешевший варіант нам не підходить, оскільки нам потрібно мінімум 2 сайти (один для React–додатку, інший для Node.js сервера). Кращий варіант коштує 285 гривень на місяць та надає 20 гігабайт пам'яті на NVMe–накопичувачі, 1 гігабайт оперативної пам'яті та можливість генерації

безкоштовного SSL–сертифікату. В цілому – прийнятний варіант для нашого вебдодатку. Найдорожчий варіант за 407 гривень надає більше пам'яті та можливість створювати безмежну кількість сайтів, але в нашому випадку це непотрібно. Зупинимось на варіанті за 285 гривень на місяць.

Висновки до розділу 3

В результаті роботи над останнім розділом, нами було спроєктовано макети сторінок вебдодатку за допомогою графічного редактору Figma. Після цього, базуючись на створених макетах, та поставлених вимогах у попередньому розділі було реалізовано фактично два додатки – сервер та клієнт. Було в повному обсязі реалізовано запланований функціонал серверної частини – створена велика кількість кінцевих точок, контролерів, схем та інших елементів коду. Створено NoSQL базу даних на основі MongoDB.

Аналогічно у повному обсязі було реалізовано клієнтський додаток, використовуючи технологію React та допоможні бібліотеки. Розроблена адміністративна панель, функціонал авторизації на основі сучасної технології JWT–токенів, створено адаптив для мобільних пристроїв. Весь вихідний код розміщено в публічних репозитаріях GitHub [49, 50]. Розроблені рішення тимчасово розміщено на безкоштовні хостинги dashboard.render та Vercel.

Проведено економічний аналіз, порахована орієнтовна вартість реалізації подібного проєкту, яка склала 38730,6 грн. Розглянуто можливість розміщення сайту на повноцінному хостингу. Для цього обрано хостинг ukraine.com.ua, та тариф «Кращий» вартістю 285 грн./місяць.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було проаналізовано велику кількість різноманітного інструментарію веброзробника. Вимоги до вебдодатків наростають надзвичайно швидко. Вони стають не просто елементом, а важливою складовою успішного бізнесу.

Проаналізовано різноманітні вебсайти та систематизовано їх за характеристиками, призначенням та оптимальним стеком технологій. В результаті дослідження можна зробити висновок, що для кожної задачі потрібен свій інструмент та підхід – універсальних рішень не існує.

Описано та порівняно frontend–рішення, серед яких HTML–препроцесори (Pug, Haml), CSS–бібліотеки (Tailwind, Ant Design та інші). Особлива увага була присвячена JavaScript–фреймворкам. Було створено порівняльні таблиці популярних фреймворків React, Angular, Vue, Svelte. Досліджено особливості використання кожного з них. Універсальним рішенням виявився React. Окремо розглядались системи керування контентом (CMS): досліджено класифікацію, особливості застосування та галузі використання, для яких вони призначені.

Також розглянуто backend–інструментарій: проведено порівняння популярних фрейморків (Laravel, Express, Django, Spring), досліджено особливості використання різних баз даних (SQL, NoSQL) та систем управління базами даних. Порівняно продуктивність та швидкодію популярних вебсерверів (Nginx, Apache). Кращі результати показав Apache. Розглянуто способи тестування та відладки серверної частини вебдодатку за допомогою спеціального програмного забезпечення (Postman, Insomnia).

Окрім цього був проведений аналіз допоміжного програмного забезпечення: системи контролю версій та хмарні репозитарії, інтегровані середовища розробки, пакетні менеджери, Docker та контейнеризація.

Проведено аналіз актуальних методологій проєктної діяльності та виділено переваги використання супровідного програмного забезпечення.

Сучасна методологія Agile є чудовим вибором для впровадження в діяльність фірми, що займається розробкою вебдодтків. В якості програмного забезпечення можна використовувати Microsoft Project/Jira для масштабних, або Asana/Trello для середніх та малих проєктів.

Головним завданням практичної частини роботи було спроєктувати та розробити вебдодаток для Навчально-дослідної лабораторії вебтехнологій та хмарних обчислень. Для цього було виконано наступні кроки:

– аналіз предметної області. Визначено призначення, основні вимоги та особливості вебсайтів закладів освіти. Створення власного вебдодатку для Навчально–дослідної лабораторії дозволить структуровано розміщувати інформацію про життя та новини лабораторії, надасть можливість всім бажаючим дізнатись про лабораторію, її керівників та ознайомитись з досягненнями студентів. Також проведена порівняльна оцінка двох існуючих вебсайтів структурних підрозділів заклаів освіти, недоліки та переваги яких було враховано під час розробки;

– обрання робочого стеку технологій. Для вибору найоптимальніших інструментів проведено багатогранний аналіз переваг та недоліків різних продуктів. В результаті було сформовано наступний стек: Figma – графічний редактор, MongoDB – база даних, Express.js – backend–фреймворк, React – frontend–фреймворк, Git – система контролю версій, GitHub – хмарний репозиторій, Visual Studio Code – редактор коду, yarn – пакетний менеджер, WebPack – система автоматизованої збірки;

– розробка макетів сторінок. Продумано та реалізовано зовнішній вигляд всіх сторінок вебдодатку, на основі яких буде створюватись сайт;

– створення серверної частину додатку. Була налаштована обробка всіх кінцевих точок, реалізована авторизація за допомогою JWT–токенів та можливість завантаження файлів;

– створення клієнтської частини додатку. На основі раніше створених макетів розроблено клієнтський додаток. Реалізована мобільна версія вебдодатку.

В результаті практичної частини було реалізовано весь запланований функціонал:

– авторизація;
– адміністративна панель;
– всі сторінки;
– додавання, редагування та видалення статей та робіт;
– можливість пошуку новин та робіт по назві;
– лічильники кількості переглядів статті.

Проведено економічний аналіз. Приблизна вартість впровадження такого проєкту становить 38730,6 грн. Досліджено варіанти розміщення вебсайту на хостингу. Зроблено вибір на користь платформи ukraine.com.ua та тарифу "Кращий" за 285 грн. на місяць.

Подальше вдосконалення роботи можливо в напрямку розвитку підтримки SEO, оскільки React дуже погано з ним взаємодіє. Також можливе розширення функціоналу. Наприклад, розробка модулю для тестування та опитування студентів, або блок з відеоматеріалами. Впровадження TypeScript в проєкт дозволить більш легко розширювати проєкт та створювати нові функціональні блоки.